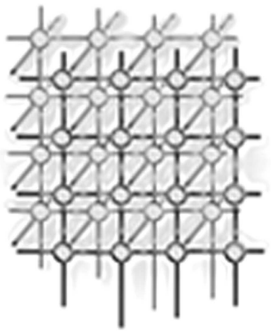
Lightweight emulation to study peer-to-peer systems

Lucas Nussbaum, Olivier Richard

CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2000; **00**:1–15 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

Lightweight emulation to study peer-to-peer systems



Lucas Nussbaum and Olivier Richard

Laboratoire d'Informatique de Grenoble - LIG
ENSIMAG - Antenne de Montbonnot
51 avenue Jean Kuntzmann
38330 Montbonnot Saint-Martin, France

SUMMARY

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 article and demonstrate the usefulness of this platform.

KEY WORDS: peer-to-peer, evaluation, emulation, network, virtualization, BitTorrent

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 accuracy of the model and the number of nodes always has to be made.

The alternate solution is to run the real application to study on a real-world experimentation platform like PlanetLab [5]. But the environment is then difficult to control and to 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 required when developing a peer-to-peer system, but it isn't enough, and developing other ways to evaluate peer-to-peer systems is becoming more important as their complexity grows [9].

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.

This paper is organized as follows. First, an overview of emulation and virtualization is provided, before related works are examined. Then, the main features of P2PLab are described and evaluated. Finally, the usefulness of P2PLab is verified by performing experiments on the BitTorrent peer-to-peer file sharing system.

EMULATION AND VIRTUALIZATION

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 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.

RELATED WORKS

A lot of work occurred recently in the virtualization area, with different approaches. Linux Vserver [14] 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 *para-virtualization* to give the ability to run simultaneously several operating systems. Those operating systems have to be modified to run over the Xen micro-kernel, 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 [16], which runs on FreeBSD and is the most popular network emulator. Dummynet is integrated into the FreeBSD firewall (`ipfw`), and is configured by

using specific firewall rules. Other tools are NISTNet [4] (which runs on Linux 2.4), and Linux 2.6's Traffic Control (TC) [17] (using NetEm [10]). Those tools schedule inbound and/or outbound packets to control bandwidth and delay, and emulate network problems such as packet loss.

Higher level typically tools use the former low-level tools to emulate complex topologies. NetBed [19] combines real nodes using RTC or DSL lines, nodes using Dummynet, and simulated nodes (using Network Simulator Emulation Layer) to provide experimental environments. Modelnet [18] 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.

P2PLAB

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 allow to run a large number of virtual nodes on each physical node. 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, and the network emulation model of P2PLab will be presented.

Suitability of FreeBSD

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 [16], 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

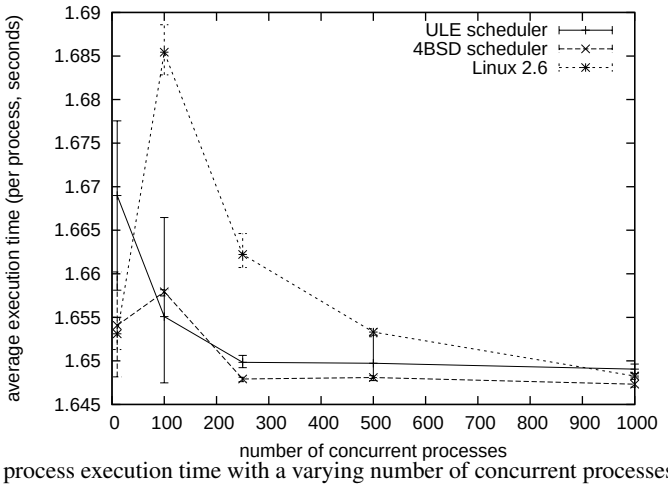


Figure 1. Average per process execution time with a varying number of concurrent processes. Processes are CPU-intensive, but not memory-intensive.

on nodes of the GridExplorer system, part of the French grid research project Grid5000 [3]. The nodes are Dual-Opteron 2 Ghz with 2 Gb of RAM and Gigabit Ethernet.

The first experiments evaluated FreeBSD's ability to run a large number of concurrent processes. In a first experiment, several instances of a non-memory-intensive program were executed (calculating Ackermann's function, requiring about 1.65 seconds to complete when run alone), and the average per process execution time was measured. Figure 1 shows that there is no overhead caused by the execution of concurrent processes. The average execution time even decreases when the number of processes increases, probably because of cache effects and costs that don't depend on the number of processes.

Then, the same experiment was done with memory-intensive processes (doing simple operations on large matrices). The results described on figure 2 are very different for Linux and FreeBSD. With Linux 2.6, the scheduler and/or the memory management prevent the execution time from increasing when all the data doesn't fit anymore in the physical memory. With FreeBSD, the execution time increases a lot as soon as virtual memory (*swap*) is used.

In the next experiments, we will have to make sure that we are in experimental conditions where virtual memory is not needed.

Fairness between processes is also important: some instances must not benefit from a larger share of the CPU time. To get a first idea of the level of fairness between processes, we perform the following experiment: we start *at the same time*[*], on the same machine, 100 instances of a same CPU-intensive

[*] An high priority process starts the instances with a lower priority. Results don't show a significant bias introduced by the start order.

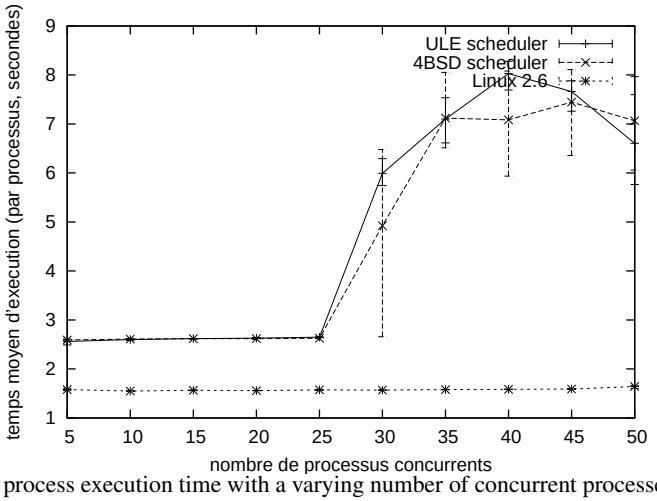


Figure 2. Average per process execution time with a varying number of concurrent processes. Processes are CPU- and memory- intensive.

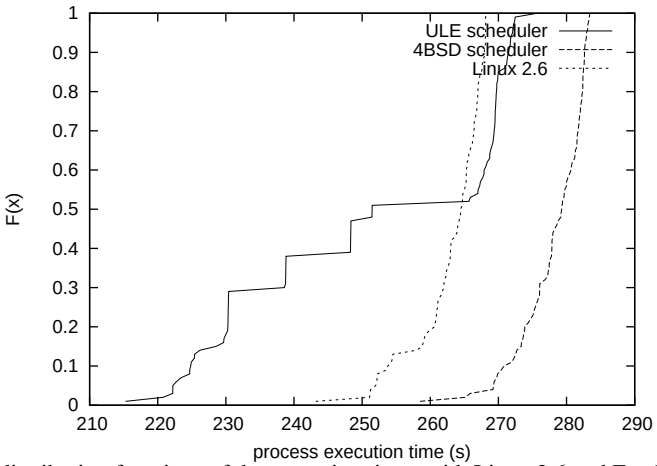


Figure 3. Cumulative distribution functions of the execution times with Linux 2.6 and FreeBSD's 4BSD and ULE schedulers.

program, and measure the execution time of each instance. When executed alone, the program needs about 5 seconds to complete.

Figure 3 shows that with the 4BSD scheduler and Linux's scheduler, most processes finish *nearly* at the same time. With the ULE scheduler, bigger variations appear. Those results are different from those obtained with FreeBSD 5 in [12], where, with the ULE scheduler, some processes were excessively privileged by the scheduler and were allowed to run alone on a CPU. This problem seems to have been fixed in FreeBSD 6.

Even if this approach provides satisfying results, both regarding scalability and fairness, it is somehow limited: it is not possible to perform experiments where virtual processors of different speeds are assigned to instances. This approach is therefore not suitable for the study of Desktop Computing systems. Using more complex virtualization solutions could help avoid this limitation, and also allow a more precise control of the memory.

In the following experiments, we used the 4BSD scheduler in P2PLab.

Virtualization

As said earlier, virtualization is made at the level of the process' network identity: instances on the same physical system share all resources (file system, 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 name-space conflicts, the addresses 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.

To bind an application to a particular IP address, we chose to intercept some network system calls (figure 5 show the order in which those system calls are used for TCP sockets). Several options were available:

- Modifying the application under study : this requires the application's source code, and the ability to recompile it.
- Linking the application with an overhead library, either at compilation-time or at run-time (using the `LD_PRELOAD` environment variable). This approach is interesting since modifying the application is not required. However, it has problems with applications which are statically compiled, or with applications using a library for network communications.
- Using `ptrace()`. This system call enables an application to monitor another one, and to intercept its system calls (User Mode Linux, for example, uses this approach). This approach requires additional system calls, causing an important overhead.
- Modifying the kernel, especially the handling of system calls. But modifying kernel code is harder and more error-prone than modifying user-level code.
- Modifying the C library (`libc`). This approach allows to intercept all system calls made by non-statically compiled applications.

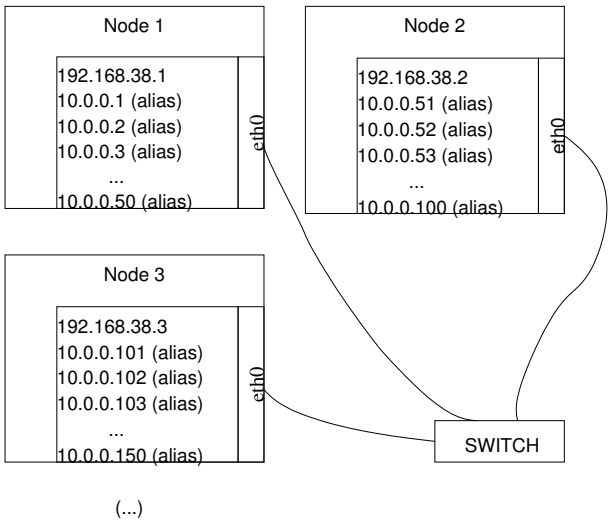


Figure 4. On each physical node, IP addresses for virtual nodes are configured as interface aliases.

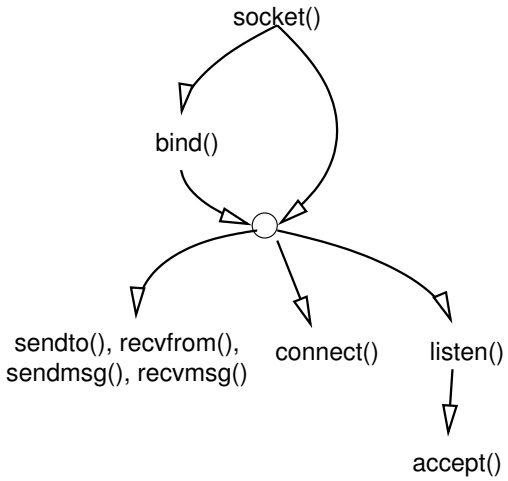


Figure 5. Network system calls used when establishing or accepting a TCP connection.

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()`. 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.

Network Emulation

Current network topology emulators like ModelNet [18] target a realistic emulation of the core network (routing and interactions between *autonomous systems*, or between the main routers). But most peer-to-peer applications run on nodes on the edge of the Internet [8]: while the traffic in the core of the Internet can influence the peer-to-peer system behavior (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, in an experimental platform like P2PLab, 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. Evaluations on P2PLab can be combined with experiments on real systems like PlanetLab [5] when an understanding of the influence of the core network is needed.

In P2PLab, we model the Internet from the point of view of the participating node, by excluding what is less important from this point of view. 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 [16], the FreeBSD traffic shaper integrated into FreeBSD's firewall (IPFW). Both incoming and outgoing packets are delayed by Dummynet. Two rules are needed for each hosted virtual node (one for incoming packets, the other one for outgoing packets).

But this model only allows to set parameters for bandwidth and latency for each virtual node. It doesn't allow to model *locality* between virtual nodes. The notion of group of nodes is therefore added, to be able to study applications using locality. In a real system, those groups would match nodes from the same ISP, from the same country, or from the same continent. P2PLab's emulation model allows to control bandwidth, latency and packet loss rate on network links between nodes and their ISP, and latency between groups of nodes, allowing to study problems involving locality of the nodes, for example.

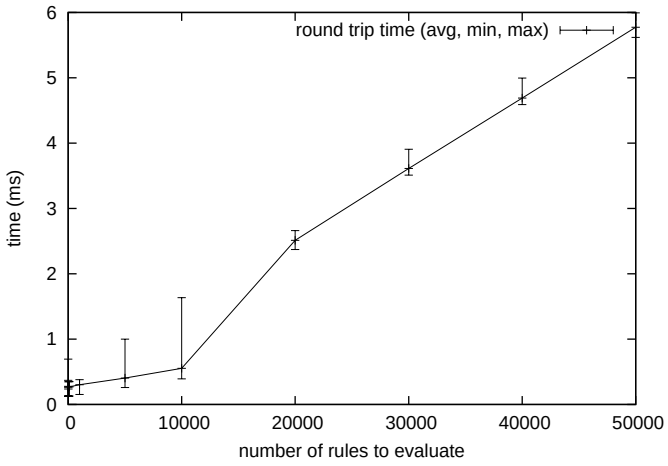


Figure 6. Measured *round-trip time* when the number of firewall rules vary.

The number of rules is the main parameter limiting the scalability of P2PLab. To determine the importance of this factor, we measure the *round-trip time* with `ping` between two nodes. When packets go out of the first node, the firewall must evaluate a varying number of rules. Figure 6 shows that latency increases nearly linearly with the number of rules, because the rules are evaluated linearly by the firewall. With IPFW, it is not possible to evaluate the rules in a hierarchical way, or with a hash table.

Figure 7 shows an example of topology that we successfully emulated with P2PLab. As an example, the physical node hosting the virtual node 10.1.3.207 will need:

- two rules for each hosted virtual node (incoming and outgoing packets) ;
- one rule to add 100ms of latency for packets coming from 10.1.3.0/24 going to 10.1.1.0/24 (the opposite rule being on the nodes hosting 10.1.1.0/24) ;
- one rule to add 100ms of latency for packets coming from 10.1.3.0/24 going to 10.1.2.0/24 ;
- one rule to add 400ms of latency for packets coming from 10.1.0.0/16 going to 10.2.0.0/16 ;
- one rule to add 600ms of latency for packets coming from 10.1.0.0/16 going to 10.3.0.0/16.

We measured the latency between nodes 10.1.3.207 and 10.2.2.117 to 853ms, which can be decomposed in:

- 20ms of delay added when the packet went out of 10.1.3.207 ;
- 400ms of delay between groups 10.1.0.0/16 and 10.2.0.0/16 ;
- 5ms of delay when the packet arrived on 10.2.2.117 ;
- 425ms for the return trip, detailed as above ;
- 3ms of overhead, because of the underlying network and the evaluation of firewall rules.

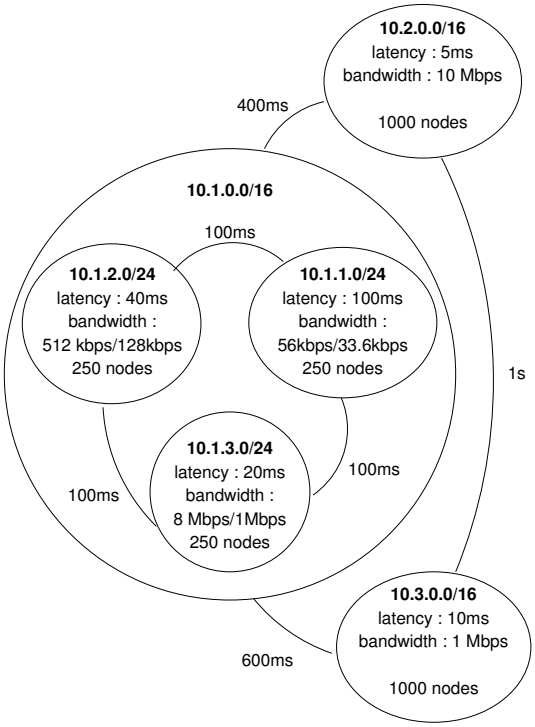


Figure 7. Emulated topology.

STUDY OF BITTORRENT WITH P2PLAB

In this section, we compare our emulation platform with BitTorrent, showing that P2PLab is a suitable experimentation platform to study complex peer-to-peer system.

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 [13, 11], analytical modeling [15] or simulation [2].

However, those works have only rarely 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 (written in Python by BitTorrent's original author) was used for all experiments. It was slightly modified to allow data collection (a time-stamp was added to the default output). The experiments took place on the GridExplorer system, already described previously.

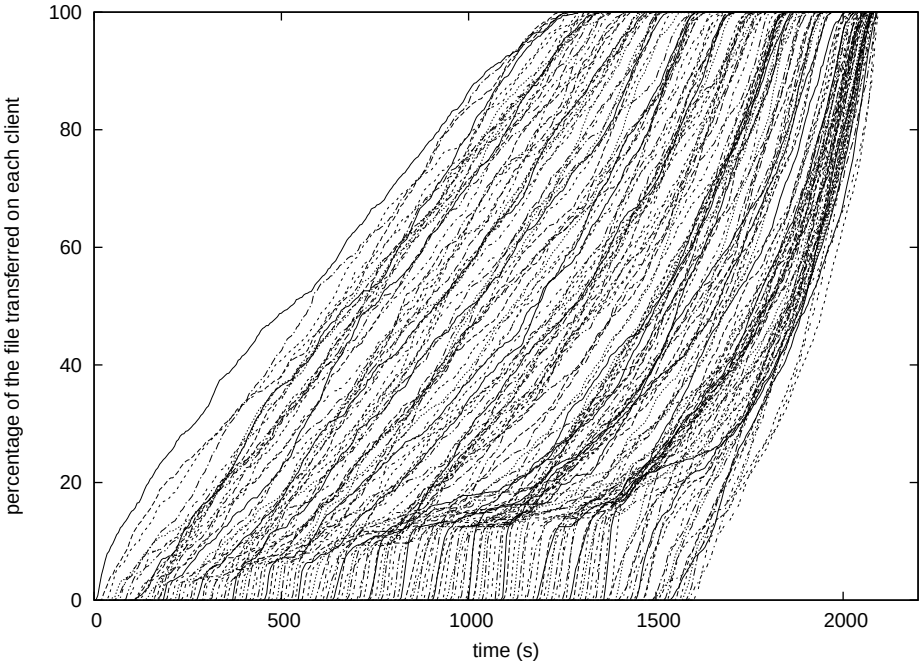


Figure 8. Evolution of the download of the 160 clients

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 10s 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:

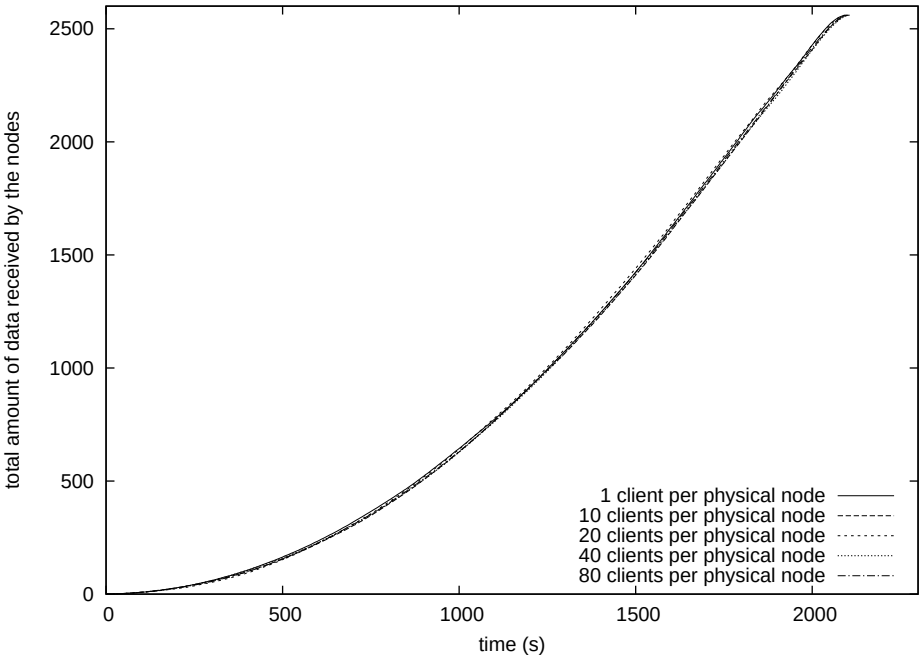


Figure 9. P2PLab folding ratio: total amount of data when downloading of a 16 MB file with 160 clients.

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

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 nodes, under the same network conditions as in

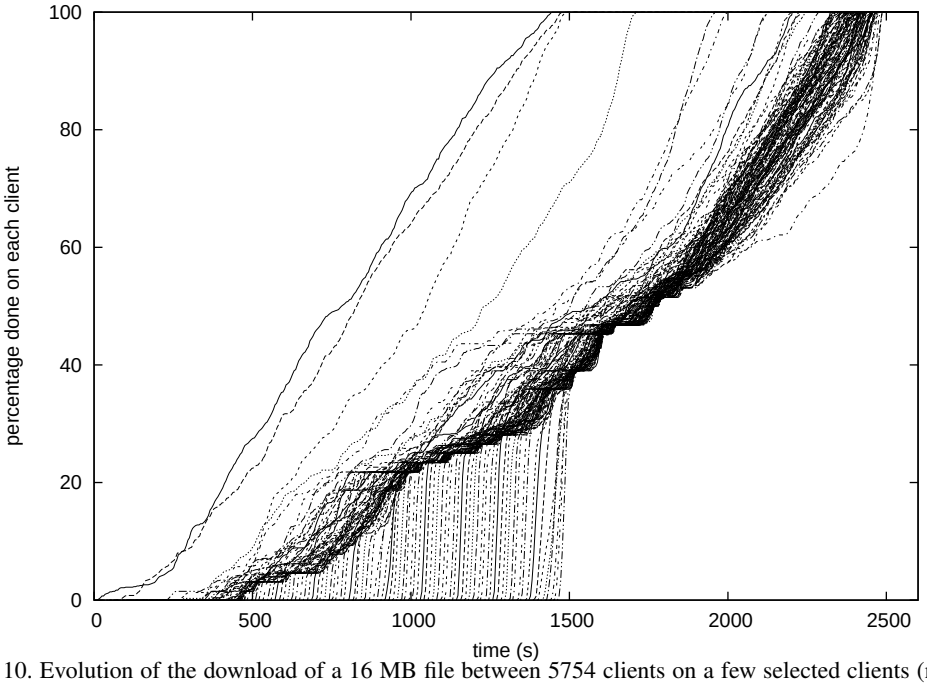


Figure 10. Evolution of the download of a 16 MB file between 5754 clients on a few selected clients (nodes 50, 100, 150, ... 5750).

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 0.25s. When a client finishes its download, it stays active and helps the others finish their download.

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

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. This paper describes P2PLab, a promising tool to study peer-to-peer systems: it allows to use the real application on a large number of nodes in a configurable environment, allowing reproduction of experiments.

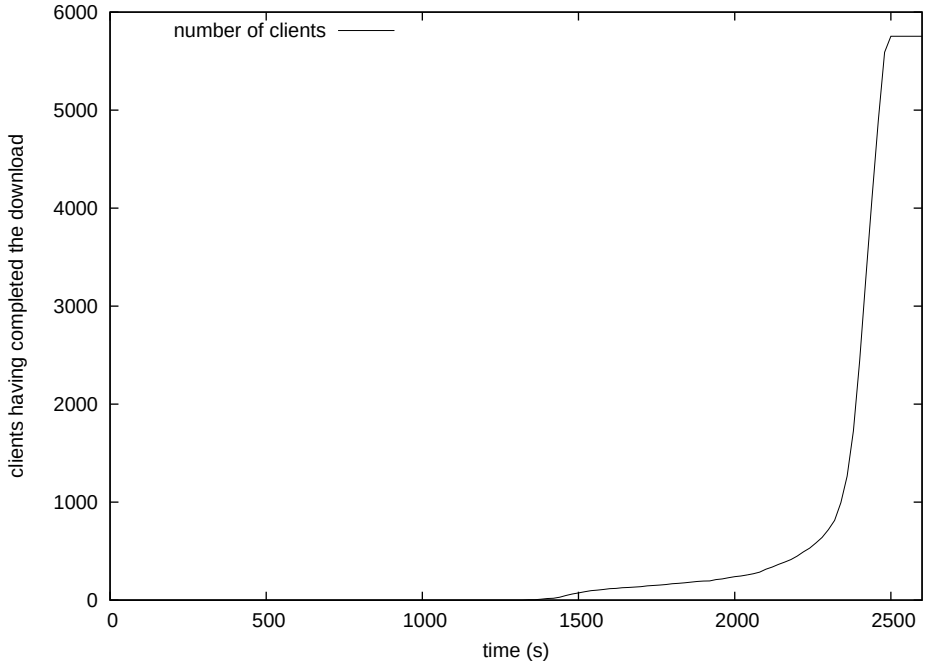


Figure 11. Number of clients having completed their download.

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.

ACKNOWLEDGEMENTS

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

REFERENCES

1. Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew 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. Ashwin R. Bharambe and Cormac Herley. Analyzing and improving BitTorrent performance. Technical Report MSR-TR-2005-03, Microsoft Research, 2005.
3. Franck Cappello, Eddy Caron, Michel Dayde, Frederic Desprez, Emmanuel Jeannot, Yvon Jegou, Stephane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, and Olivier 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. Mark Carson and Darrin Santay. NIST Net: a Linux-based network emulation tool. *SIGCOMM Comput. Commun. Rev.*, 33(3):111–126, 2003.
5. Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, July 2003.
6. Bram Cohen. Incentives build robustness in BitTorrent. http://www.bittorrent.com, 2003.
7. Jeff 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. Ian T. Foster and 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.
9. Andreas Haeberlen, Alan Mislove, Ansley Post, and Peter Druschel. Fallacies in evaluating decentralized systems. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS'06)*, February 2006.
10. Stephen Hemminger. Network emulation with NetEm. In *linux.conf.au*, 2005.
11. Mikel Izal, Guillaume Urvoy-Keller, Ernst W Biersack, Pascal A Felber, Anwar Al Hamra, and 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.
12. Lucas Nussbaum and 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.
13. J.A. Pouwelse, P. Garbacki, D.H.J. 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.
14. The Linux VServer Project. http://www.linux-vserver.org/Linux-VServer-Paper, 2003.
15. Dongyu 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.
16. Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997.
17. Linux Advanced Routing and Traffic Control. http://lartc.org/.
18. Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostić, Jeff Chase, and David Becker. Scalability and accuracy in a large-scale network emulator. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 271–284, New York, NY, USA, 2002. ACM Press.
19. Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270, 2002.