Virtualization Techniques in Network Emulation Systems

Roberto Canonico, Pasquale Di Gennaro, Vittorio Manetti, and Giorgio Ventre

Dipartimento di Informatica e Sistemistica
Università di Napoli Federico II
via Claudio 21, 80125 Napoli, Italy
{roberto.canonico,pasquale.digennaro,
vittorio.manetti,giorgio}@unina.it

Abstract. The continuous increase of computational power has made viable the implementation of more and more sophisticated virtualization techniques. The use of virtualization in cluster environments to build on-demand computing infrastructures is a recent trend with a great potential. Cluster-based network emulators are a specific class of cluster-based systems whose main purpose is to help researchers evaluate the effectiveness of new protocols and applications in realistic, synthetically generated network scenarios. Both large scale experimental testbeds (such as PlanetLab) and cluster-based network emulation systems (such as Emulab) use virtualization techniques at the basis of their resource management mechanisms to achieve isolation and concurrent experiments execution. In this paper, we compare different virtualization techniques already adopted in this kind of distributed systems and illustrate the peculiar virtualization requirements of a cluster-based network emulator. Furthermore, we show how Xen can be used to build a flexible and scalable network emulation system.

1 Introduction

In the last few years, network emulation has gained interest in the community of network reserchers, being considered an important technique to evaluate the effectiveness of new protocols and applications in heterogeneous, controllable and realistic network scenarios. In a network emulation experiment, simulated network elements interact, in real-time, with real network components and applications. Today's most complex network emulation systems are cluster-based. These systems are made of a large number of hardware components arranged in a common facility that can be remotely accessed by users through a web interface. Components include links, switches, routers, and PCs that interchangeably play the roles of end-systems, routers, or WAN emulators. An efficient use of the available hardware resources is one of the main goals, usually achieved by means of a combination of virtualization and space-sharing that aims at allowing simultaneous non-interfering experiments.

In a typical cluster-based network emulation system, users submit to the system an experiment request. An experiment request contains a "virtual" network

L. Bougé et al. (Eds.): Euro-Par 2007 Workshops, LNCS 4854, pp. 144–153, 2008.

description to be reproduced with the available cluster resources. A conservative resource allocation policy consists in mapping the emulated "virtual" nodes onto dedicated PCs and emulated links onto switched ethernet links. Nowadays, with increasing computational power made available at low-cost, it is possible to exploit virtualization techniques to map multiple "virtual" nodes on a single CPU. There are many good reasons for doing that. For example many applications need to be evaluated on large topologies, yet they are not resource hungry. Moreover, multiplexing provides a more efficient use of communication resources as the bandwidth of the emulated geographic links is usually much less than that available in the local interconnect used in a modern cluster [1]. These reasons motivate the use of small-scale clusters to emulate medium/large size topologies in a inexpensive manner [2],[3].

In the rest of this paper we will illustrate the importance of virtualization for network emulation. We compare different virtualization techniques already adopted in cluster-based network emulation systems. Furthermore, we show how Xen can be used to build a flexible and scalable network emulation system.

2 Cluster-Based Network Emulation Systems

Maybe the most complex cluster-based emulation system developed so far is Emulab [4]. Emulab is a free-for-use, Web-accessible, time- and space-shared, reconfigurable network testbed, providing integrated access to a wide range of experimental environments. The Emulab core consists of several hundred rack-mounted PCs, combined with secure, user-friendly web-based tools, and driven by ns-compatible scripts or a Java GUI, allowing remote configuration and control of machines. Even the OS of a cluster node can be fully and securely replaced with custom images by any experimenter.

In Emulab an enhanced version of FreeBSD jail has been used, which allows the creation of isolated environments (vnodes), characterized by independent namespaces. Each of these vnodes is accessible not only through the host node, but also remotely via ssh. This kind of virtualization technology used by Emulab is not resource hungry: this gives the opportunity to build a relatively large number of vnodes even on not very powerful machines. Anyway, this mechanism does not offer fully isolated execution environments, potentially creating some security issues. For this reason, in Emulab all vnodes running on a given physical host must belong to the same experiment. Even the network is not completely virtualized, since much of the network stack is shared between physical host and vnodes.

The Network Emulation Testbed project [5] provides a configurable network environment for the performance analysis of distributed applications and protocols, consisting of a 64 node PC cluster system running Linux connected by a flexible network infrastructure. The network infrastructure can be set up in arbitrary ways, emulating anything from Wide Area Networks (WANs) to highly dynamic Mobile AdHoc Networks (MANETs). The node PCs are connected by means of a Gigabit Ethernet switch on which an arbitrary number of VLANs

can be configured. Through the use of VLANs, nodes can be connected with any possible virtual topology. Network traffic on emulated links is controlled by a special traffic shaper module, called NETshaper, implemented as a Linux kernel module. NETShaper provides a link-layer emulation that is completely transparent to upper layers. Parameters that can be emulated by NETShaper include fixed delay, variable delay, and frame loss.

NEPTUNE is a cluster-based emulation system developed at University of Napoli. Even though many design assumptions made for NEPTUNE were borrowed by Emulab, since from the early stages of design, NEPTUNE has assumed virtualization as a key technology for realizing complex networking scenario. The NEPTUNE project was started in 2004 at University of Napoli. The project main goal is to create a cluster-based network emulation system that could be used to assess either new networking technologies and protocols (e.g. to test new QoS Routing protocols and Traffic Engineering schemes in MPLS-based networks), as well as new distributed applications (e.g. multimedia peer-to-peer applications).

At the time of this writing, the NEPTUNE emulation system runs on a cluster of workstations consisting of 28 biprocessor nodes ProLiant DL380, each equipped with two Intel Pentium IV Xeon 2.8 GHz CPUs, 5 GB of PC-2100 RAM, one 100 Mbps Ethernet NIC, one Gigabit Ethernet NIC. Each node is equipped with a 34.6 GB SCSI disk. A 700GB centralized disk array is also available to the whole cluster. The cluster nodes are connected each other through a set of 100/1000 Ethernet switches.

One of the cluster nodes, the NeptuneManager, provides the fundamental services (like dhcp, dns, tftp, nfs, and so on) needed to properly configure at boot-time the physical cluster nodes and the virtual machines participating to the emulation experiments. A web-based system is used to manage and configure the whole system.

Setting up an emulation experiment in NEPTUNE consists primarily in defining a "virtual topology" made of emulated intermediate network nodes (routers) and end-system nodes (user terminals). A complex networked system can be reproduced by allocating multiple "virtual" network nodes (both routers and end systems) on each of the cluster physical nodes. A testbed mapping module (much like the one used in Emulab [6]) is responsible of mapping the "virtual" topology onto the cluster physical resources. Virtual network nodes are implemented in NEPTUNE as Xen virtual machines. The main advantage of the use of virtualization techniques to instantiate virtual network nodes is the significant reduction in equipment and management costs. Virtual machines allow the creation of customized execution environments, where customization consists in selecting the operating system, installed software packages and user access policies. Furthermore, virtual machines can be paused or shut down at any time, and later resumed, even at a different physical location (migration). Finally, virtual machines support fine-grained mechanism for resource usage control, allowing to define (and even change at run-time) precise limits to the the amount of usable RAM and disk space.

3 Virtualization Technologies

Virtualization is a widely used technique in which a software layer multiplexes lower-level resources among higher-level software programs and systems.

In a non virtualized system, a single OS controls all hardware platform resources. A virtualized system includes a new layer of software, the virtual machine monitor (VMM). A virtual machine monitor manages the creation, destruction and control of one or more virtual machines (VM) on a computer, and is responsible for controlling access to the resource of the real hardware, as well as multiplexing the execution of multiple VMs fairly. Virtual machines do not access the system's real resources directly, but through the VMM.

Some virtualization techniques support migration of virtual machines. In addition to facilitating hardware maintenance operations, VM migration can be triggered automatically by workload balancing or failure-prediction agents.

In the following we will present a few virtualization technologies that have been used in distributed experimental infrastructures to support multiple concurrent experiments.

3.1 FreeBSD Jails

The Emulab system supports multiple experiments running concurrently on the same physical node. This is implemented thanks to the use of the FreeBSD Jail mechanism. The FreeBSD Jail facility provides the ability to partition the operating system environment. Administrators can create several independent mini-systems called jails and provide access to the super-user account in each of these without losing control of the over-all environment. Each jail is a virtual environment running on the host machine with its own files, processes, user and superuser accounts. From within a jailed process, the environment is indistinguishable from a real system. A process in a partition is referred to as in jail. When a FreeBSD system is booted up after a fresh install, no processes will be in jail. When a process is placed in a jail, it and any descendents of the process created after the jail creation, will be in the same jail. A process may be in only one jail, and processes within the jail are prevented from delivering signals to processes outside the jail. The only way for a new process to enter the jail is by inheriting access to the jail from another process already in that jail. Processes may never leave the jail they created, or were created in. When a jail is created, it is bound to a particular file system root. Processes are unable to manipulate files that they cannot address, and as such the integrity and confidentiality of files outside of the jail file system root are protected. Security is simply guaranteed because the jail environment is separated from the rest of the system, in other words, since the jail is limited to a narrow scope, the effects of a misconfiguration or mistake does not jeopardize the rest of the system's integrity. Modifying the running kernel by direct access and loading modules is prohibited, just like modifying the network configuration and accessing raw, divert and routing sockets are prohibited. Thanks to the limited scope of a jail, it allows administrators to

painlessly delegate several tasks which require superuser access without handing out complete control over the system. With jails it is possible to install different daemons in different jails and delegate their administration to other people by giving them access to the superuser account. It is safe because the jailed superuser has limited privileges and he can't escape the jail because he cannot get any information about the base system. Virtualization is valuable to service providers wishing to offer their users the ability to have custom configurations and yet keep the overall system easy to maintain.

3.2 Linux VServer

The Linux-VServer technology implements a soft partitioning concept based on Security Contexts which permits the creation of many independent Virtual Private Servers (VPS) running simultaneously on a single physical server. A VPS provides an almost identical operating environment as a conventional Linux server. All services can be started on such a VPS, without modification, or with only minimal modifications. The implementation of Security Contexts requires some modification to the plain Linux kernel. The purpose of a Context is to hide all processes outside of its scope, and prohibit any unwanted interaction between a process inside the context and a process belonging to another context. This separation requires the extension of some existing data structures in order for them to become aware of contexts and to differentiate between identical uids used in different virtual servers. It also requires the definition of a default context that is used when the host system is booted, and to work around the issues resulting from some false assumptions made by some user-space tools that the init process has to exist and to be running under id '1'.

The real drawback when VServer is used in a network emulation system, is that networking is based on isolation, not on virtualization. This prevents each virtual server from creating its own internal routing or firewalling setup. Furthermore, it is not possible to assign different MAC addresses to VPS.

3.3 OpenVZ

OpenVZ [7] is an another operating system-level virtualization technology built using GNU/Linux. It gives the ability to run multiple isolated system instances, called *Virtual Private Servers (VPS)* or *Virtual Environments (VE)*. It does not offer the same flexibility in the choice of the operating system, if compared to other solutions such as VMware and Xen, but in many usage scenarios it can be an interesting solution. Networking in OpenVZ is implemented through a virtual device (*venet*). Network emulation can also benefit of the use of Virtual Ethernet device (*veth*), that is an Ethernet-like device which can be instantiated inside a VE. A veth can be assigned a MAC address and used in a bridged configuration, emulating a sort of "virtual switch" inside the host, to which all virtual interfaces created in the VEs are connected. Each veth can be configured via dhcp at boot time, when the VE is started.

3.4 Xen

Xen [8] is a paravirtualization system developed by the University of Cambridge. Xen provides a virtual machine monitor for x86 processors that supports execution of multiple guest operating systems at the same time.

Today, a special Xen-compatible version of Linux, XenoLinux, is available. According to Xen researchers, 100 XenoLinux instances can be run simultaneously on a single Xen VMM with minimal performance degradation. Xen-compatible version of Windows XP and NetBSD are actively being developed at the time of this writing.

4 Virtualization for Node Multiplexing

Node multiplexing is the problem of emulating more than a network node on the same physical cluster node. This problem is inherently a problem of machine virtualization, as it has been described in the previous section. Hence, it can be solved with one of the many available virtualization technics. Aspect to be taken int account to select the proper one for a a cluster-based emulation system are efficiency, scalability, flexibility, isolation, and operating system customization.

In Emulab, an extended version of FreeBSD Jails is used. Jail allows the creation of different execution environments at the same time. The user can perform remote ssh to each of the execution environments. The Jails implementation is relatively lightweight, so it is possible to instantiate several execution environments on the same machine. Unfortunately, the Jails approach has also some drawbacks. In particular, since it does not rely on a virtual machine monitor, the degree of isolation among different execution environments is limited. To limits the negative effects of this problem, in Emulab, all the execution environments activated on the same physical machine must belong to the same emulation experiment. Furthermore, the communication resources are not virtualized at all: the whole protocol stack is shared among the various execution environments. This, incidentally, make it impossible for Jail to support communication with guaranteed Quality of Service. Finally, the Jail mechanism is currently only available in the FreeBSD kernel, and its porting to other platforms does not appear straightforward.

The node multiplexing technique we chose for NEPTUNE is Xen, due to its many advantages, and in particular because Xen is:

- highly scalable;
- potentially supports different kinds of Operating Systems;
- provides good isolation among different virtual machines running concurrently;
- supports virtual machine migration, allowing dynamic re-allocation of experiments on the cluster nodes;

and also because Xen implements different optimization techniques in the communication mechanisms, allowing good communication performance among virtual machines implemented within the same physical node.

5 Virtualization for Link Multiplexing

The nodes of a cluster are connected by means of one or more switched Ethernet LANs. Each cluster node may be equipped with one or more (Giga or Fast) Ethernet NIC. These NICs, in turn, may be connected to the same switch or to different switches. In theory, it would be possible to connect the cluster nodes in pairs, by means of crossed Ethernet cables, so to physically reproduce the desired topology. However, such a solution is not viable, for at least two reasons. Firstly, because changing the network topology would be extremely impractical, time-consuming and error-prone. Secondly, because this would make it impossible to emulate network topologies with a number of links greater than half of the number of Ethernet NICs. Hence, practical solutions require to emulate multiple point-to-point connections on top of one or more shared Ethernet LANs. This is usually performed by means of Virtual LANs (VLANs) [1], [5]. Such a solution is implemented by properly configuring the Ethernet switches and does not require any configuration and processing in the cluster nodes. This makes, however, the system configuration software extremely dependent on the characteristics of the network switches. For the above reasons, we decided not to use VLANs in NEPTUNE and we adopted two network device independent solutions for link multiplexing:

- IP-aliasing and destination MAC address filtering
- Virtual NICs

The first technique is our choice when the emulated node has been mapped directly onto a physical node, i.e. there is no node multiplexing. The other solution consists in activating a virtual NIC and binding it to one virtual interface of a virtual machine. Traffic shaping is obtained by means of queuing disciplines directly attached to the virtual interfaces. Main advantages of this technique is a more clean management of the networking during the experiment creation and the possibility to use classless shapers in addition to classful ones, which are required to emulate characteristics of emulated links when ip aliasing is used.

6 Virtualization Techniques for Link Multiplexing Compared

In this section we show a comparison of two network emulation solutions, implemented by means of two different virtualization techniques both supporting the creation of a virtual interface inside a VM, namely OpenVZ and Xen. OpenVZ is an example of an operating system-level server virtualization solution, while Xen is an example of an hypervisor based on the paravirtualization concept.

Our emulated network consists of two end systems interconnected by means of a couple of intermediate IP routers. We compare two different implementations of this emulated network: one in which routers are implemented in Xen VMs,

and another in which routers are implemented in OpenVZ VMs. In both cases we allocated the two VMs in two different Linux boxes. We also implemented a reference scenario, in which the intermediate routers are plain Linux boxes. The three scenarios are depicted in 1.

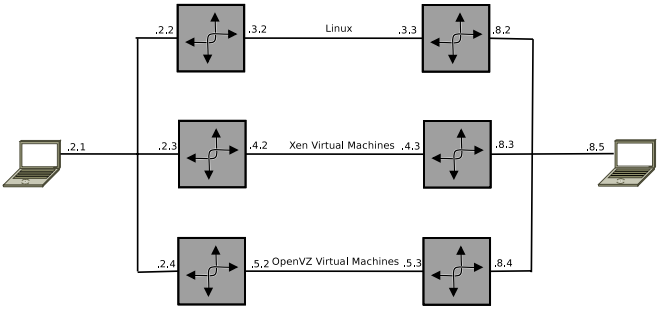

Fig. 1. Our experimental testbed

Physical machines have an identical hardware configuration: each of them is a SMP system with two Intel Xeon running at 2.8Ghz and is equipped with 5GB of RAM. For our tests we have used only one Tigon 3 gigabit ethernet interface on each machine. Virtualization of the network is obtained by means of IP-aliasing. In Xen we have created for each virtual router two virtual NICs. Xen networking has been configured in the bridge configuration: in dom0 a bridge has been created to which all virtual interfaces are connected. OpenVZ offers two networking implementations: *virtual ethernet* and *virtual network*. We have chosen the first one, as it gives the opportunity to assign MAC addresses to virtual interfaces.

Constant bit rate traffic has been generated between end hosts by use of D-ITG traffic generator [9], varying packet size and inter-departure time (IDT). In figure 2 we show packets dropped at increasing inter-departure rates; one Kbyte packets were used. Results demonstrate that for trasmission rates not exceeding 30000 packets per second, we almost don't have losses. Reducing inter-departure times, causes an increasing drop rate. Drop rates are always higher in Xen as compared to OpenVZ. Anyway, in both virtualization solutions, losses reduce the sustainable effective throughput in such configurations.

The mean jitter measured at the receiver node (estimated on 10 milliseconds intervals) has been plotted in figures 3, 4, 5 for the three scenarios . Since the experiment has been conducted in absence of any other interfering traffic, jitter is entirely due to the variable packet processing time in the intermediate routers. This experiment has been conducted by generating CBR streams made of 1KB packets transmitted at an increasing rate. Each stream has been generated continuously for 30 seconds. The comparison of results shows that OpenVZ routers introduce, on average, 50% less jitter than Xen. Similar results have been noticed varying packets generation rates.

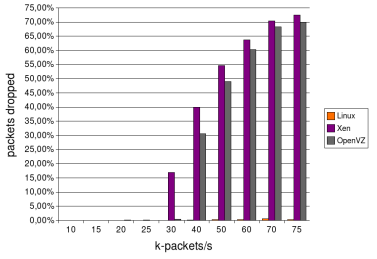

Fig. 2. Packets dropped vs. tx rate

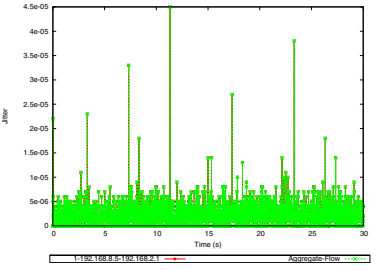

Fig. 3. jitter introduced by Linux routers

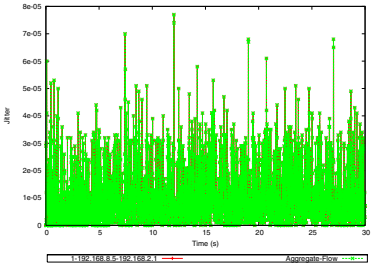

Fig. 4. jitter introduced by Xen routers

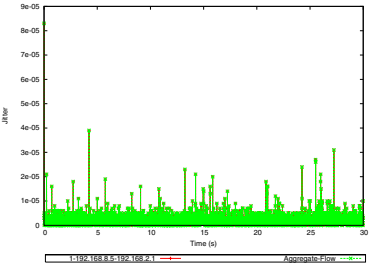

Fig. 5. jitter introduced by OpenVZ routers

7 Conclusions

Virtualization is maybe the most important tool to design effective network emulation systems. Virtualization of resources, in fact, offers two fundamental benefits: first, it allows to allocate more than one network node per single physical node; secondly, it is an instrument to allocate the available computational resources of a cluster among different users and different experiments, in an on-demand manner. When it comes to network emulation, it is not only important to virtualize CPUs, but also communication resources, e.g. network interfaces and their available bandwidth. For this reason, the designers of a cluster-based network emulation system need to take this specific requirement into account when they select the virtualization technique to be used at the foundation of their system. Our tests show that most of today's virtualization techniques have some problems when they have to deal with huge amount of network traffic. Our preliminary results show that OpenVZ performs better than Xen. Nonetheless, Xen is largely more easy to use and more flexible, and this is a great advantage for cluster-based systems of several tens or hundreds of machines. Furthermore, the fault-isolation capabilities of paravirtualization systems also play in favor of Xen. For all the above reasons, we decided to base the NEPTUNE emulation

system on Xen, being confident that most of today's limitations will be overcome in future releases.

Acknowledgements

This work has been partially funded by the FP6 IST-2005-034819 project *OneLab: an open networking laboratory supporting communication network research across heterogeneous environments*. Pasquale Di Gennaro has been funded from the Italian Ministry of University and Research (MIUR), in the form of a grant in the framework of the S.Co.P.E. Project.

References

1. Hibler, M., Ricci, R., Stoller, L., Duerig, J., Guruprasad, S., Stack, T., Webb, K., Lepreau, J.: Feedback-directed virtualization techniques for scalable network experimentation. Technical report, University of Utah, Flux Group Technical Note 2004-02 (May 2004)
2. Guruprasad, S., Ricci, R., Lepreau, J.: Integrated network experimentation using simulation and emulation. In: Proceedings of TridentCom, IEEE, Los Alamitos (2005)
3. Maier, S., Herrscher, D., Rothermel, K.: On Node Virtualization for Scalable Network Emulation. In: Proceedings of the 2005 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2005), Philadelphia, PA, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Simulation Councils, Inc., July 24–28, 2005, pp. 917–928 (2005)
4. White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., Joglekar, A.: An integrated experimental environment for distributed systems and networks. In: Proc. of the Fifth Symposium on Operating Systems Design and Implementation, pp. 255–270. USENIX Association, Boston (2002)
5. Herrscher, D., Leonhardi, A., Rothermel, K.: On node virtualization for scalable network emulation. In: Proceedings of the 2005 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2005) (July 2005)
6. Ricci, R., Alfeld, C., Lepreau, J.: A solver for the network testbed mapping problem. SIGCOMM Comput. Commun. Rev. 33(2), 65–81 (2003)
7. OpenVZ server virtualization open-source project, `http://openvz.org`
8. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: Procs. of the 19th ACM Symposium on Operating Systems Principles, SOSP 2003, pp. 164–177. ACM Press, New York (2003)
9. Avallone, S., Emma, D., Pescapè, A., Ventre, G.: Performance evaluation of an open distributed platform for realistic traffic generation. Performance Evaluation (Elsevier) 60(1-4), 359–392 (2005)