A KVM-Based Tool for Evaluating and Enhancing Virtual Routing Performance

Luca Abeni
University of Trento
luca.abeni@unitn.it

Csaba Kiraly
University of Trento
kiraly@disi.unitn.it

This paper describes a new tool to build virtual routers, allowing to easily perform reproducible performance tests with different virtual routing architectures and different software versions and configurations. Some usage examples are described, showing how the presented tool allows to compare the performance of various virtual router implementations, and can be used for research on virtual routing.

1 Introduction

With the advent of cloud computing, outsourcing and virtualization of computing resources has became a huge commercial success in recent years. As more and more complex systems are being outsourced, virtualization of networking resources has also become a hot topic investigated by researchers. Although software routers have been available for some time, virtualization of software routing resources brings now opportunities, such as supporting hot migration of routers in case of hardware of link failures [WKB+08], providing fine-grained router level control of shared hardware resources to virtual network operators [SKHL12], or the virtualization of complex distributed systems as a whole.

It has been shown [FIM+11] that recent advances in CPU, chipset and NIC architectures and, in general, advances in off-the-shelf commodity hardware enable high performance software routers. However, building high performance virtual routers remains challenging, as the number of architectural choices and fine-tuning parameters is vast, and knowledge about the effect of these choices on performance is still rather limited. Moreover, experimenting with various options could be very time consuming due to the time required to generate virtual machines with various configurations and to run a reasonable set of performance benchmarks.

In fact, results reported in literature are hardly comparable to each other due to the large variety of virtualization technologies, operating system kernel versions, and networking configurations used.

In [EGH+08, EGH+10] authors analyse performance characteristics of software routers as well as of Virtualised software routers. Both Xen and OpenVZ

based virtualization are studied, and various possible system architectures are confronted. They conclude that the effect of architectural choices is tremendous, observing differences of an order of magnitude.

[RHS11] and [RHS10] focuses on various ways of conveying packets between the host OS and guest virtual routers (OpenVZ and Linux Namespaces are their preferred virtualization platform). The performance gain achieved by using the macvlan device is studied and modifications to macvlan are proposed. Similarly, [BHEKP11] works on improving performance of passing packets to VRs, but their modification groups packets and conveys them in batches to avoid I/O overhead.

Authors of [BMM+08] build their own container-based network virtualization solution. Performance tests show results that are better than those of Xen or OpenVZ based solutions.

[SKHL10] defines an interesting algorithm for increasing scalability in the number of VRs that could be run by the same physical host, however, performance of the proposed scheme is evaluated only in simulation, without a real implementation.

Although several of the above papers show more-or-less detailed performance results, reproduction of these is cumbersome due to the many undocumented minor details. Unfortunately, these minor details could cause tremendous performance differences. For example, Xen based solutions are sometimes shown as having really bad performance, while in other cases Xen performs quite well. Moreover, to the best of our knowledge, none of the above systems have been packaged and published in a way that simplifies comparison with other approaches.

This paper contributes to research on virtual routing by introducing an open source tool for building virtual routers, and running reproducible (and comparable) performance tests. The presented approach allows to easily test and compare different virtual routing technologies and different software versions and configurations, and to repeat the same experiments (same virtual router configuration, etc...) on different machines.

2 The Problem

When implementing a *Virtual Router* (VR), there are many different possibilities and implementation options. For example, a software router can be modified to implement multiple virtual forwarding tables [WKB+08], or an unmodified software router can be executed inside a *Virtual Machine* (VM). In the second case, a program called *Virtual Machine Monitor* (VMM) provides a virtualised environment in which the software router is executed. The VMM generally executes in a *host* Operating System (OS), while the software executing inside the VM is called *guest* (hence, in this context the VR is just a specialised guest). The VMM can provide either "OS level" (container-based) virtualisation [lxc, ope], full hardware virtualisation [Bel05], or some form of paravirtualisation [BDF+03, Rus08].

Even when focusing on unmodified software routers (virtualised by executing them in a VM) and when considering only one VM type, there still are numerous different implementation options. For example, the routing/forwarding subsystem of an OS kernel (such as Linux) can be used (running the OS in a VM that uses full hardware virtualisation or paravirtualisation), or some different kind of (kernel space or user space) software router, such as click [KMC+00] can be used instead. Moreover, the virtual router can be monolithic or can be split in different modules to improve performance or flexibility [AKLB]. Each one of these different design decisions implies performance/flexibility trade-offs, and from a research perspective it can be useful to compare the routing performance of the various options. This kind of experimentation requires to set up several similar VR implementations, only changing some well defined details that are under investigation, and to repeat exactly the same experiment (without changing other variables) for each setup in order to get comparable results.

Moreover, the VMM and host kernel version and configuration can heavily affect the virtual routing performance. Hence, once the VR structure and design is fixed different versions and configurations of the software need to be tested and compared. As a result, for each VR setup it is necessary to generate and install different versions and configurations of the host tools.

Finally, performance tests have to be performed on different machines, and need to be reproducible. Hence, some way to reproduce exactly the same host and guest configuration on a number of different nodes is needed. An approach that is often used to run experiments with virtual routers is to install a general purpose host OS, to install the VMM on such a general purpose OS, and then to configure the host OS and the VMM. However, this might require an excessive amount of time (and disk space): for example, installing a Linux distribution on a PC might require almost 1GB of disk space and a considerable time (about 30 minutes). This approach also requires to manually configure all the nodes involved in a test, and to copy large virtual router images (sometimes, up to 4GB of disk image).

3 Building the VR and the Host

The performance tests described in Section 2 generally require the following steps to be performed:

guest image generation First of all, the routing software has to be installed and configured in a guest image, in order to setup a VR. In the simplest case, this simply requires to install a guest kernel and some routing daemon (monolithic router), in other cases, some additional software (for example, the Click modular router) has to be installed and configured. Note that in some situations, the additional routing software might depend on the OS kernels: for example, Click can be compiled as a Linux kernel module, but only supports Linux kernels up to 3.0. Hence, a specific kernel version has to be installed in the guest, and the Click modules have to be compiled for that specific kernel

host installation and configuration A VMM has to be installed in the host system, together with the guest image generated in the previous item, together with all its dependencies

VMM and virtual networking setup The VMM has to be configured to properly start one or more VRs. This also requires to setup some kind of virtual network in the host (to connect the virtual network interfaces, as seen by VR, to the physical interfaces of the host, or to connect the various guests in case of modular VRs). This can be done using different technologies (for example, TAP or MACVTAP interfaces, vhost-net accelerations in case of paravirtualised network interfaces, etc...).

Two software packages have been developed to automate the steps mentioned above: `VRouterScripts` and `ImageScripts`.

`VRouterScripts` allows to easily setup KVM-based VMs and the virtual networks needed to connect them and to create a VR (step 3).

`ImageScripts` build the VR, taking care of the guest image generation, and install the VR and a VMM in the host (steps 1 and 2). Basically, this package compiles, installs and configures different kinds of Linux kernels and routing software in the guest image, and compiles, installs and configures KVM to execute such a generated guest image. The `VRouterScripts` is then used to control KVM and to start the VR. All the generated binaries are installed (together with `VRouterScripts`) in a small Linux-based host OS in a bootable USB key. Hence, the host does not need to be installed on the test PC, but can be booted from USB.

This approach allows to generate pre-configured bootable host images that contain everything needed to run virtual routing experiments. The version and configuration of the VMM and host kernel are decided when building the image, and no other tuning, configuration or installation step is needed. Note that the VR is also configured, built, and installed in the bootable image, so that it automatically starts at boot time, without needing other configurations.

The generated images are also optimised for size, so that they can be installed on small USB keys, and can be easily moved between different machines. A configured guest fits in less than $16MB$ of disk space, and a configured host image (containing a simple virtual router) fits in $64MB$. This result has been obtained by using the Linux `initramfs` technology to store the core of the OS (all the binaries, configuration files and scripts needed to boot and to start a shell), and using traditional disk partitions only to store additional components (in the host, the VMM, `VRouterScripts`, and the guest image; in the guest, the routing software). The compressed "OS core" is based on busybox[1] and almost the same for host and guest (only the kernel modules contained in it are different). Its size is about $8MB$.

The flexibility of `VRouterScripts` allows to easily test different technologies for interconnecting VRs with the host's physical interfaces (or for interconnecting VRs running on the same host, or different modules of a modular VR),

[1] `http://www.busybox.net`

ranging from more "traditional" TAP interfaces (which do not provide good performance) to experimental technologies such as netmap [Riz12]. In this way, it has been possible to evaluate the performances of the various solutions, and to compare them. Similarly, `ImageScripts` allows to install different versions of the Linux kernel, and to compile it with different options. This allowed, for example, to compare the performance of a 64bit host kernel and a 32bit host kernel (see Section 4).

All the software is open-source (under the GNU General Public License - GPL), and a first release will be available soon[2] and is designed and implemented to be portable, and have no particular dependencies. A generic x86 machine with the Linux distribution of choice can be used to build an host image, only requiring to have a C compiler and few basic development libraries installed. This result is obtained by recompiling from source all of the software installed in the host[3].

4 Usage Examples

To show the usefulness of the presented tools, some simple examples of its usage are presented. All the examples are based on tests that have been performed using `ImageScripts` and `VRouterSctipts` to compare the routing performance of various VRs and various software configurations. Notice that all the setups used in these experiments are based on `VRouterSctipts` and have been automatically generated by `ImageScripts`.

First of all, `ImageScripts` and `VRouterSctipts` have been used to setup a *monolithic* VR (the routing subsystem of a a 3.2.4 Linux kernel running inside a VM) and to evaluate its performance when changing some configuration parameters in the host kernel and in the VMM. The goal of this set of experiments is to show how the proposed tool help in testing different VMM and host configurations and versions for the same kind of VR (in this case, the monolithic VR). Figure 1 reports the performance of the monolithic VR running on a 32bit host kernel and on a 64 bit host kernel. The results for the 32bit host kernel have been measured when using only 1 CPU core, 2 CPU cores, or 4 CPU cores. When 4 CPU cores have been used, the VMM has been executed with and without static binding of the VMM threads to the CPU cores. From the figure, it is possible to notice that the routing performance improves when increasing the number of CPU cores used; moreover, static binding of the various VMM threads to the CPU cores can further improve the performance (since migration overhead is removed). Finally, a 64bit kernel (using the `x86_64` ISA) using 4 cores with static CPU bindings provide the best performance. This is probably due to the fact that the `x86_64` architecture provides more general purpose CPU registers, thus the kernel can be more efficient.

[2] currently, a preliminary alpha version is available - contact the authors via email if interested

[3] Note that this is also a simple way to respect the GPL.

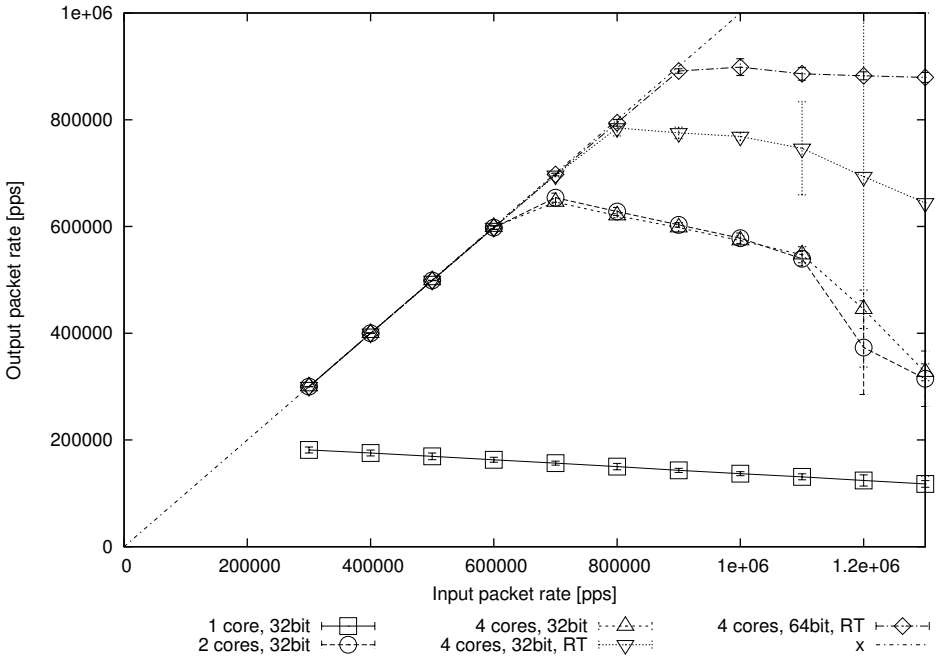

Figure 1: Performance of a monolithic VR, with different numbers of CPU cores and different configurations.

The proposed approach can also be used to compare the performance of different VR designs. For example, Figure 2 compares the Parallel Virtual Routing (PVR) architecture [AKLB] with the monolithic VR already evaluated in Figure 1 (the best performing monolithic configuration is used in this example). The monolithic VR's performance curve is topped at 900 Kpps. Even in the best case, with 4 cores and proper binding, it starts to loose packets as soon as the offered load increases above this level. Using the PVR architecture, cycles of additional CPU's are better exploited. The PVR architecture can achieve higher forwarding rates (both when using 2 and when using 3 VRs in parallel), although some losses can be observed here as well above 900 Kpps.

Finally, Figure 3 shows the results of some experiments performed with a more complex VR architecture: the MultiStage Router (MSR) [BBGL10]. Again, these experiments compare the performance of different VMM configurations (since the MSR is modular and the different modules execute in different VMs, the configuration space is much larger than for a simple monolithic VR). The figure shows how the presented tool can be used to tune the VMM configuration for performance, by evaluating the impact of various configuration parameters on the MSR routing performance. There are significant differences between results obtained without binding (allowing VM processes to migrate between CPUs and relying only on the scheduler to handle the CPU allocation)

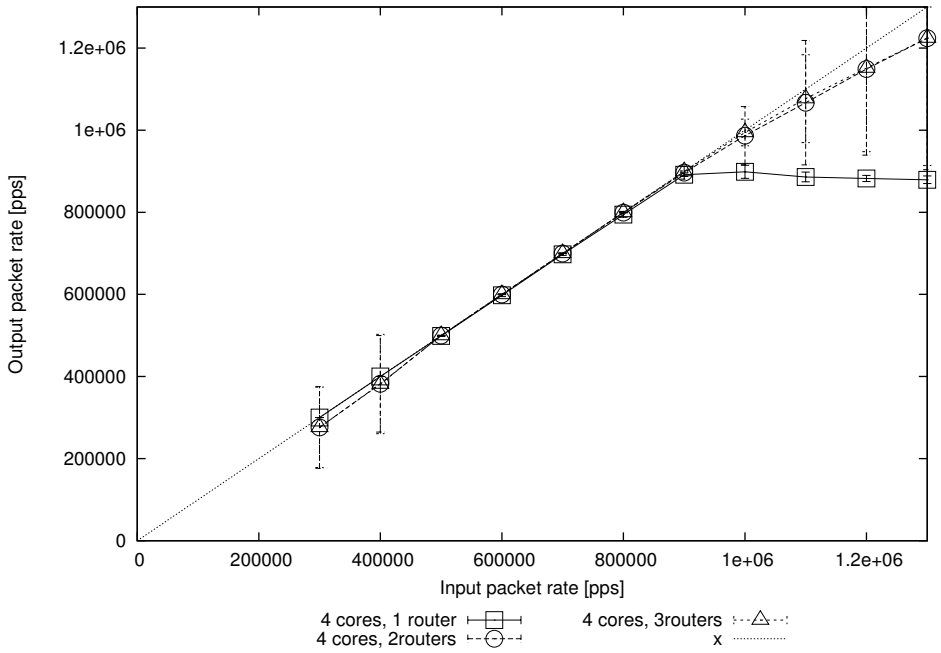

Figure 2: Comparison between the performance of a monolithic VR and the performance of a PVR.

and with various CPU bindings. By selecting the proper binding, the performance curve can be optimized for various targets, like statistical stability (low statistical variance between test runs), or overload stability (flatter performance curve in case of high offered traffic).

Notice that performance is in general lower in this case than in the case of a monolithic or PVR virtual router achitecure, which is due to the improved flexibility provided by the MSR architecture.

5 Conclusions and Future Work

This paper described `ImageScripts` and `VRouterScripts`, two software packages that can be used to simplify the automatic creation of VRs. These tools are useful for research because they allow to easily reproduce performance experiments in a predictable way, and to build different implementations and version of a VR, in order to understand the performance impacts of the various design decisions.

As a future work, the software will be extended to support OS-level virtualisation (based on LXC) and some research technologies (such as real-time kernels, reservation-based scheduling, new drivers architecture, etc...) in order

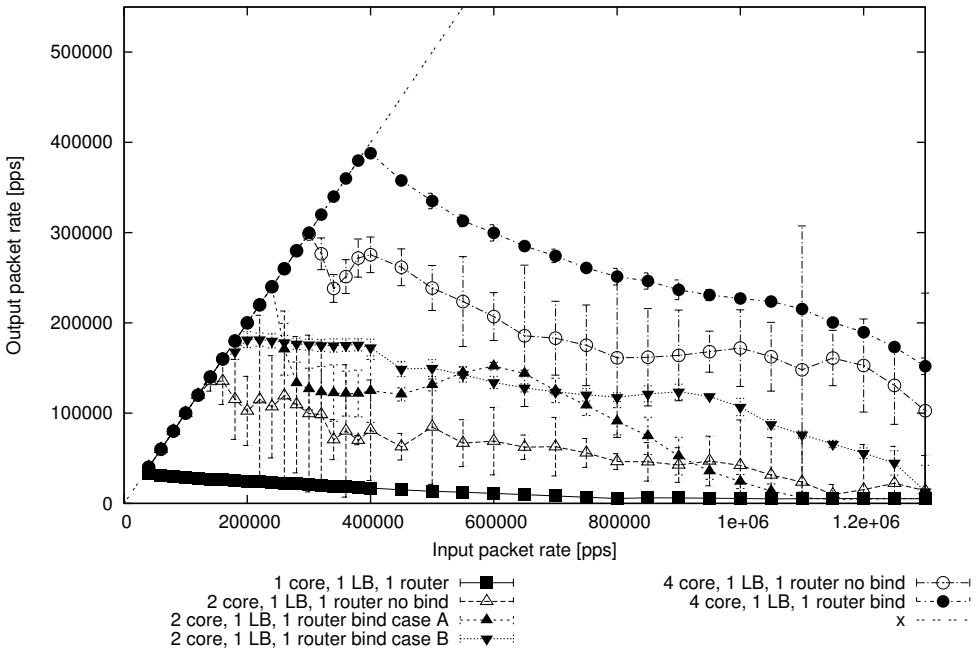

Figure 3: Performance of an MSR with different configurations.

to evaluate their performance. It will also be used to build a more complex testbed for evaluating virtual routing performance in more realistic situations. The usage of KVM migration capabilities to migrate the VRs between different hosts will also be investigated.

References

[AKLB] Luca Abeni, Csaba Kiraly, Nanfang Li, and Andrea Bianco. Tuning kvm to enhance virtual routing performance. In submission at ICC 2013.

[BBGL10] Andrea Bianco, Robert Birke, Luca Giraudo, and Nanfang Li. Multistage software routers in a virtual environment. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM 2010)*, Miami, Florida, December 2010.

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebar, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP03)*, 2003.

[Bel05] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *Proceedings of the 2005 USENIX Annual Technical Conference*, Anaheim, CA, April 2005.

[BHEKP11] M. Bourguiba, K. Haddadou, I. El Korbi, and G. Pujolle. A container-based i/o for virtual routers: Experimental and analytical evaluations. In *Communications (ICC), 2011 IEEE International Conference on*, pages 1 –6, june 2011.

[BMM+08] Sapan Bhatia, Murtaza Motiwala, Wolfgang Muhlbauer, Yogesh Mundada, Vytautas Valancius, Andy Bavier, Nick Feamster, Larry Peterson, and Jennifer Rexford. Trellis: a platform for building flexible, fast virtual networks on commodity hardware. In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, pages 72:1–72:6, New York, NY, USA, 2008. ACM.

[EGH+08] Norbert Egi, Adam Greenhalgh, Mark Handley, Mickael Hoerdt, Felipe Huici, and Laurent Mathy. Towards high performance virtual routers on commodity hardware. In *Proceedings of the 2008 ACM CoNEXT Conference*, CoNEXT '08, pages 20:1–20:12, New York, NY, USA, 2008. ACM.

[EGH+10] Norbert Egi, Adam Greenhalgh, Mark Handley, Mickael Hoerdt, Felipe Huici, Laurent Mathy, and Panagiotis Papadimitriou. A platform for high performance and flexible virtual routers on commodity hardware. *SIGCOMM Comput. Commun. Rev.*, 40(1):127–128, January 2010.

[FIM+11] Kevin Fall, Gianluca Iannaccone, Maziar Manesh, Sylvia Ratnasamy, Katerina Argyraki, Mihai Dobrescu, and Norbert Egi. Routebricks: enabling general purpose network infrastructure. *SIGOPS Oper. Syst. Rev.*, 45(1):112–125, February 2011.

[KMC+00] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297, 2000.

[lxc] `http://lxc.sf.net`.

[ope] Openvz linux containers. http://www.openvz.org.

[RHS10] M.S. Rathore, M. Hidell, and P. Sjodin. Performance evaluation of open virtual routers. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 288 –293, dec. 2010.

[RHS11] MuhammadSiraj Rathore, Markus Hidell, and Peter Sjdin. Data plane optimization in open virtual routers. In Jordi Domingo-Pascual, Pietro Manzoni, Sergio Palazzo, Ana Pont, and Caterina Scoglio, editors, *NETWORKING 2011*, volume 6640 of *Lecture*

Notes in Computer Science, pages 379–392. Springer Berlin Heidelberg, 2011.

[Riz12] Luigi Rizzo. Revisiting network I/O apis: the netmap framework. *Communications of the ACM*, 55(3):45–51, Mar 2012.

[Rus08] Rusty Russel. virtio: Towards a de-facto standard for virtual I/O devices. *ACM SIGOPS Operating Systems Review*, 42(5), 2008.

[SKHL10] Haoyu Song, Murali Kodialam, Fang Hao, and T. V. Lakshman. Building scalable virtual routers with trie braiding. In *Proceedings of the 29th conference on Information communications*, INFOCOM'10, pages 1442–1450, Piscataway, NJ, USA, 2010. IEEE Press.

[SKHL12] Haoyu Song, M. Kodialam, Fang Hao, and T.V. Lakshman. Efficient trie braiding in scalable virtual routers. *Networking, IEEE/ACM Transactions on*, 20(5):1489 –1500, oct. 2012.

[WKB+08] Yi Wang, Eric Keller, Brian Biskeborn, Jacobus van der Merwe, and Jennifer Rexford. Virtual routers on the move: live router migration as a network-management primitive. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 231–242, New York, NY, USA, 2008. ACM.