Computer Communications 70 (2015) 40–53

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

On the performanc of KVM-based virtual routers

Luca Abeni [a,*], Csaba Kiraly [b], Nanfang Li [c], Andrea Bianco [d]

[a] DISI – University of Trento, Via Sommarive 5, Trento (TN), Italy
[b] Bruno Kessler Foundation, Via Sommarive 18, Trento (TN), Italy
[c] Embrane, Inc., 2350 Mission College Blvd, Santa Clara, CA 95054, USA
[d] Dipartimento di Elettronica e delle Telecomunicazioni, Politecnico di Torino, Torino, Italy

ARTICLE INFO

Article history:
Received 31 March 2014
Revised 13 May 2015
Accepted 16 May 2015
Available online 29 May 2015

Keywords:
Software routers
Virtualisation
Data plane performance

ABSTRACT

This paper presents an extensive experimental evaluation of the layer 3 packet forwarding performance of virtual software routers based on the Linux kernel and the KVM virtual machine. The impact of various tuning and configuration options on forwarding performance is evaluated, focussing on the mechanism used for moving data to and from virtual machines, the algorithm used for scheduling the virtual router tasks, the number of used CPU cores, and the router tasks affinities. The presented results show how to properly configure the virtual router components to improve forwarding performance and the benefits of using appropriate CPU schedulers. Furthermore, some advanced architectures based on virtual router aggregation are evaluated. The presented experiments show that architectures based on router aggregation can better exploit the available CPU cores to reach performance not far from the ones obtained by non-virtualised software routers.

1. Introduction

Software routers (SRs), *i.e.*, routers implemented by software running on commodity off-the-shelf hardware, became in recent years an appealing solution compared to traditional routing devices based on custom hardware. SRs' main advantages include cost (the multi-vendor hardware used by SRs can be cheap, while custom equipments are more expensive and imply higher training investment), openness (SRs can be based on open-source software, and thus make use of a large number of existing applications) and flexibility. Since the forwarding performance provided by SRs has historically been an obstacle to their deployment in production networks, recent research works focused on increasing SRs performance by either using massively parallel hardware such as a GPU to process packets [1], allowing the routing software to directly access the networking hardware (thus eliminating the overhead introduced by the OS kernel) [2], or using other similar techniques to improve the forwarding performance of monolithic routers. An orthogonal approach to improve SR performance can be based on the aggregation of multiple devices to form a more powerful routing unit like the multistage software router [3], Router Bricks [4], and DROP [5]. While by improving the performance of a single routing device it is possible to reach the forwarding speed of multiple tens of Gigabit per second [1], the aggregation of multiple routing units can allow the forwarding speed to scale almost linearly with the number of used devices [3].

As recognised by several researchers [6,7], virtualisation techniques could become an asset in networking technologies, improving SRs flexibility and simplifying their management. As an interesting example, the virtual machines (VMs) live migration capability could be adopted for consolidation purposes and/or to save energy. Moreover, running a SR in a VM allows to dynamically adapt the forwarding performance of the (virtual) device to the workload by renting virtual resources instead of buying new hardware. This feature is especially useful when the network traffic has a high variance, thus a high processing power might be necessary only for short periods. Virtualisation can also simplify the SR management, and improve its reliability: for example, VMs migration during maintenance periods can be implemented, and faster reaction to failures should be expected by booting new VMs on general purpose servers. Finally, virtualisation improves hardware efficiency because the same physical infrastructure can be sliced and shared among different tenants.

Obviously, the usage of virtual software routers (VSRs) might increase the complexity of the routing software: for example, the communications between VMs and the physical nodes hosting them result in complex interactions between hardware and VMs, which could easily compromise VSR's performance. This paper focuses on analysing such interactions to identify and remove various performance bottlenecks in the implementation of a VSR. Since the knowledge of the behaviour of software components (routing software, virtual machine software, operating system kernel, etc.) and of their interaction mechanisms makes this investigation easier, this work

* Corresponding author. Tel.: +39 0461 28 1516.
E-mail addresses: luca.abeni@unitn.it (L. Abeni), kiraly@fbk.eu (C. Kiraly), nanfang@embrane.com (N. Li), andrea.bianco@polito.it (A. Bianco).

http://dx.doi.org/10.1016/j.comcom.2015.05.005
0140-3664/© 2015 Elsevier B.V. All rights reserved.

focuses on an open-source virtualisation environment (KVM, the Kernel-based Virtual Machine [8]) which permits to easily analyse the VSR behaviour to identify performance bottlenecks. Indeed, since KVM is tightly integrated into Linux from 2.6.20 on, all the available Linux management tools can be exploited.

VSR performance can be improved by carefully tuning various system parameters such as those related to the mechanisms used to move data to and from VMs, threads priorities and CPU affinities. Some preliminary results [9] show that a proper configuration and optimisation of the virtual routing architecture and the aggregation of multiple VSRs (as suggested by the multistage software router architecture [3]) permit to forward 64 bytes packets at about 1200 kpps in a commodity PC, close to the physical speed of a Gigabit Ethernet link. This paper extends the preliminary results starting from a detailed analysis of the packets forwarding path, which is used to develop a large number of new experiments to investigate the bottlenecks that should be bypassed to reach the full line rate. Some of the new experiments investigate the different technologies that can be used to interface the VMs with the physical hardware, comparing their performance and CPU usage. Then, the impact of the CPU scheduler on the VSR performance are investigated, showing how a proper scheduling algorithm (with properly tuned scheduling parameters) permits to control the performance of a VSR and to limit its CPU usage.

The reminder of this paper is organised as follows: Section 2 briefly summarises the virtualisation technologies used in this paper. Then, Section 3 introduces the so-called *monolithic VSR*, gives a basic evaluation of its performance, and shows that the scalability of a VSR is not affected by virtualisation. Based on these results, Section 4 discusses many optimisation techniques to tune the performance of a monolithic VSR. Section 5 discusses the problem of clustering multiple VSRs into a single logical routing unit to improve the performance or the flexibility of a VSR. Some state of the art related works are presented in Section 6. Finally, Section 7 concludes the paper. Appendix A shows the detailed packet forwarding lifecycle inside the VSR.

2. Virtualisation technologies

A SR operates both on the data plane (or forwarding plane), where each packet is handled individually and forwarded towards the next hop IP router, and on the control plane, where routing tables are filled based on routing protocol interactions. While some VSRs virtualise only the control plane and directly configure a non-virtualised data plane, in this work we concentrate on VSRs that virtualise both planes.

VSR performance is mainly affected by the amount of computational resources (*i.e.,* CPU power) available on the physical node that hosts the VSR. Such computational resources are mainly used to provide data plane operations by:

1. the forwarding/routing code in the SR (named as *guest* because it runs *inside* a VM).
2. the physical machine hosting the VM (named as *host*), to move packets among physical interfaces, virtual switches, and guest virtual interfaces.

Control plane operations are less CPU intensive and have less stringent timing constraints. Therefore, we concentrate on the evaluation of the above described data plane operations.

Several packet processing functions may be available in SRs. Thus, the amount of CPU time needed to process packets inside the SR may vary significantly. We can identify SRs used in the access network, close to end users, typically executing several functions, or SRs in the core network that mainly focus on high forwarding performance. In the access, many high layer (4–7) functionalities could be executed in the SR, including NAT, VPN encryption/decryption, packet filtering based on different rule sets, etc. Things become even more complex when QoS comes to the picture. These operations may require several CPU cycles, because different SR subsystems need to access and modify the packets. As a result, the packet processing overhead increases, decreasing the throughput. On the other hand, routers which only implement layer 3 forwarding are often used in core networks and in data centres. In this case, packets traverse only a simple and high-performance forwarding subsystem (that can run in the Linux kernel, in the Click software, or in other specific software modules). Obviously, high performance and fast forwarding speed is the key index to measure the quality of such VSR. This paper focuses on the study of VSR layer 3 forwarding, showing that a VSR can achieve almost line rate in forwarding if appropriate optimisation techniques are used.

Moving packets between the host and the VM can be executed in the OS kernel, in a hypervisor, or in a user-space component (typically, the virtual machine monitor – VMM), depending on the specific virtualisation architecture. This operation, which is crucial for the VSR performance, can be performed using different software components. This paper considers three different approaches, namely `macvtap`, bridge (plus tap), and `netmap`, analysing their performance in details.

If the VSR is implemented using a "closed" virtualisation architecture such as VMWare [10], it is not easy to understand how much CPU time is consumed by the VMM, by the guest, or by the host OS kernel. Hence, in this paper an open-source virtualisation architecture is used. The two obvious candidates are Xen [11] and KVM [8]. Since the KVM architecture is more similar to the standard Linux architecture, it has been selected for running the experiments presented in this paper. KVM is based on i) a kernel module, which exploits the virtualisation features provided by modern CPUs to directly execute guest code, and ii) a user-space VMM, based on QEMU [12], which virtualises the hardware devices and implements some virtual networking features.

The most relevant feature for VSRs provided by the user-space VMM is the emulation of network interfaces, because CPU virtualisation is not an issue, as KVM allows guest machine instructions to run at almost-native speed. When a packet is received, the VMM reads it from a device file (typically the endpoint of a TAP device) and inserts it in the ring buffer of the emulated network card (the opposite happens when sending packets). When emulating a standard network interface (such as an Intel e1000 card), the VMM moves packets to/from the guest by emulating all hardware details of a real network card. This process is time consuming, easily causing poor forwarding performance, especially when considering small packets, and/or high interrupt rates. This issue can be addressed by using `virtio-net`,[1] which does not emulate real hardware but uses a special software interface to communicate with the guest (that needs special `virtio-net` drivers). Thus, the overhead introduced by emulating networking hardware is reduced, and forwarding performance is improved. The para-virtualised NIC is based on a ring of buffers shared between the guest and the VMM, which can be used for sending/receiving packets. The guest and the VMM notify (to each other) when buffers are empty/full, and the `virtio-net` mechanism is designed to minimise the amount of host/guest interactions (by clustering the notifications, and enabling data transfer in batches).

When using `virtio-net`, the user-space VMM is still responsible for moving data between the (endpoint of the) TAP interfaces and the `virtio-net` ring buffers. Hence, when a packet is received, as depicted in the left hand side of Fig. 1:

1. the host kernel notifies the user-space VMM that a new packet is available on the TAP device file;

[1] `virtio-net` is a para-virtualised I/O framework for high speed guest networking [13].