Electronic Communications of the EASST
Volume 17 (2009)

Workshops der Wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen 2009 (WowKiVS 2009)

Designing a Platform for Flexible and Performant Virtual Routers on Commodity Hardware

Norbert Egi, Adam Greenhalgh, Mark Handley, Mickaël Hoerdt, Felipe Huici, Laurent Mathy and Panagiotis Papadimitriou

4 pages

Guest Editors: M. Wagner, D. Hogrefe, K. Geihs, K. David
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Designing a Platform for Flexible and Performant Virtual Routers on Commodity Hardware

Norbert Egi[1], Adam Greenhalgh[2], Mark Handley[3], Mickaël Hoerdt[4], Felipe Huici[5], Laurent Mathy[6] and Panagiotis Papadimitriou[7]

[1] n.egi@lancaster.ac.uk [4] m.hoerdt@lancaster.ac.uk
[6] l.mathy@lancaster.ac.uk [7] p.papadimitriou@lancaster.ac.uk
Computing Dept., Lancaster University, UK

[2] a.greenhalgh@cs.ucl.ac.uk [3] m.handley@cs.ucl.ac.uk
Dept. of Computer Science, University College London, UK

[5] felipe.huici@nw.neclab.eu
NEC Europe, Heidelberg, Germany

Abstract: Multi-core CPUs, along with recent advances in memory and buses, render commodity hardware a strong candidate for software router virtualization. In this context, we present the design of a new platform for virtual routers on x86 hardware. We also elaborate on our design choices in order to achieve both high performance and flexibility for packet processing.

Keywords: virtualization, router, platform design, commodity hardware

1 Introduction

Recent years have seen numerous software prototypes that aimed to explore the potential of router virtualization (e.g., [5]). Although these virtual router prototypes were PC-based implementations, none of them intended to exploit the recent advances in commodity hardware, such as multi-core CPUs, in order to achieve high performance. On the contrary, they were mainly focused on functionality. Modern PC architectures offer a significant level of flexibility and performance combined with a wealth of open-source software. This means that they are readily usable in order to build a virtual software router platform. Such flexibility is usually not available from commercial router virtualization solutions.

Using a modern multi-core PC, we showed that it is able to forward minimum-sized packets at a respectable aggregated throughput of 3.6Gb/s [2]. The limiting factor for the throughput is the memory latency of the uniform memory hierarchy found in most modern PCs. This leaves plenty of spare CPU cycles which can be used to run a number of virtual router instances concurrently on a single box. Is it possible to have the best of both worlds, that is to build a flexible virtual router which minimize the virtualization overhead ? This challenge is certainly more demanding for a virtual router platform than for a conventional server virtualization platform because of the short-lived nature of packets inside a router.

In [1] we tackled basic fairness issues and limitations of a modern PC for software packet forwarding, exploring alternative virtualization technologies and different forwarding scenarios.

From these findings we derived the main design decisions for our virtual router platform, which has the following salient features:

- Highly configurable forwarding planes for advanced programmability.
- Optimized CPU core scheduling for high performance.
- Hardware multi-queueing for sharing interfaces between virtual routers.

We present a high-level overview of this design which aims to answer the questions raised in the following section.

2 Virtual Router Platform Design

Our virtual router platform leverages modern and future hardware trends to provide high packet forwarding rates and advanced programmability for the forwarding engines. The support of different virtual forwarding scenarios allows to adjust the trade-off between performance, flexibility, isolation, and fairness. The platform comprises the following basic components (Fig. 1):

- A privileged domain for the management of guest domains.
- An isolated driver domain (IDD) for aggregated packet forwarding.
- A number of guest domains which host control planes (one per virtual router) and may also include a forwarding path when increased isolation and safety properties are required.

We increase the stability of the platform by moving the forwarding functionality into a new forwarding domain, decoupling it from the management domain. The forwarding domain (i.e., the IDD) can always be restarted by the management domain which is also responsible for the instantiation, termination and monitoring of the various guest domains and the IDD.

The IDD is a virtual machine which has devices mapped in to it and runs the device drivers (which, in our case, are the network devices and drivers) and typically hosts the forwarding planes. The IDD hosts the merged forwarding path and provides the ability to control and configure the individual forwarding paths (FPs) to the guest domains. The merging process enables the consolidation of all FPs in the IDD, allowing a large number of virtual routers to share common interfaces. In [2] we showed that forwarding within a common domain results in much higher performance than forwarding in the separate guest domains by avoiding the costly hypervisor domain switches per packets. Fig. 2 depicts this configuration with FP1 being off-loaded from the second guest domain into the IDD.

The platform supports two additional packet forwarding configurations :

- Splitting a FP between the IDD and a separate guest domain while using I/O channels for inter-domain communication (e.g., FP2 and FP3 in Fig. 1).
- Mapping interfaces directly into guest domains such that each FP resides in a separate guest domain (e.g., FP4 in Fig. 1).

These last two configurations enable us to safely run forwarding paths composed of untrusted Click [4] elements without compromising the performance or safety of the FPs in the IDD.

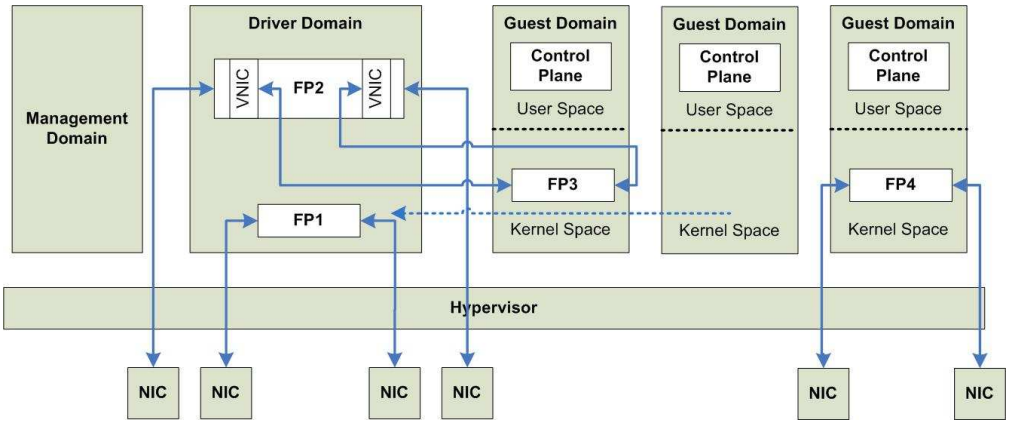

Figure 1: Platform overview.

The control planes for the virtual routers reside in the guest domains (Fig. 1). The platform supports off-the-shelf control plane solutions, such as *XORP*[3], which run in user space. Offloading the forwarding paths from a specific guest domain leaves only the control plane residing in it (e.g., the second guest domain in Fig. 1). In every other case, the guest domain includes one or multiple FPs in the kernel space.

3 Packet Processing Scheduling

In order to optimize the performance of the platform we need to take advantage of both the multi-core CPUs and exploit their cache hierarchies to minimise the effects of the concurrent memory accesses latency bottleneck[2]. We reduce the number of accesses to main memory by keeping the packets within the CPU caches as they are being forwarded. Costly memory accesses can be prevented by either keeping the packets within the same CPU core or using two CPU cores that share the same L2 cache. Our core scheduling strategy is based on these findings and observations. We briefly describe how we assign FPs to cores in order to avoid performance degradation when all FPs are consolidated in the IDD.

For a virtual router composed of k input interfaces and p output interfaces, we set up $k * p$ forwarding paths that keep an incoming packet on the same CPU core independently of the look-up decision. We therefore consider a virtual queue for every possible FP, between its input and output interfaces; Fig. 2 illustrates an example with 2 pairs of interfaces and 4 FPs. For each FP, we essentially need to schedule two tasks: (i) the task that polls packets from the input interface or *Poll Device* (PD), and (ii) the task that writes packets to the output interface or *To Device* (TD). Both tasks for each FP should be scheduled in the same core to prevent packets from switching cores. Hence, we allocate separate cores per input interface.

A critical issue for a virtual router platform is how to share interfaces between virtual routers. In [2] we showed that simple software de-multiplexing cannot guarantee fairness, especially in the presence of prioritization in the routers. In order to provide fairness, we enable hardware packet classification on the network interface cards, enabling packets to be filtered into different queues which can be subsequently polled by different virtual routers. The number of supported queues is currently restricted to 16 when filtering on the MAC level. However, future hard-

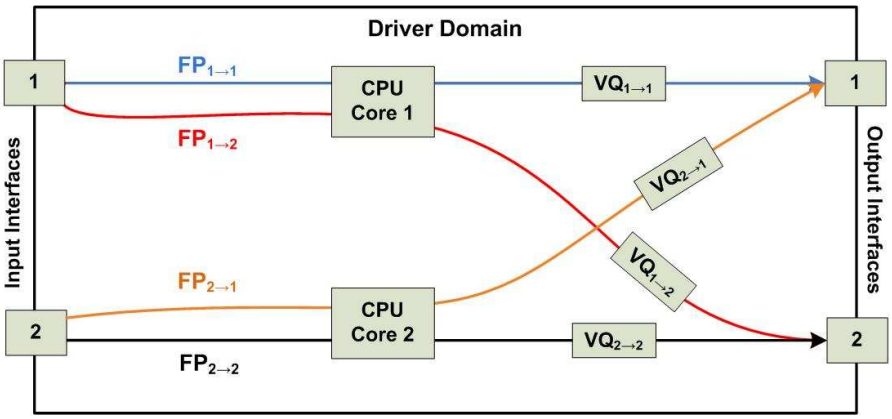

Figure 2: Scheduling forwarding paths to CPU cores.

ware trends are likely to overcome this limitation by offering more virtual queues per physical interface.

4 Conclusions

We presented an overview of our platform design for flexible and performance virtual routers based on commodity hardware. Our preliminary performance studies reveal high packet forwarding rates which, combined with the flexibility afforded by general-purpose processors, confirm the efficiency of the platform.

Bibliography

[1] Norbert Egi, Adam Greenhalgh, Mark Handley, Mickael Hoerdt, Felipe Huici, and Laurent Mathy. Fairness issues in software virtual routers. In *Proceedings of PRESTO'08*, Seattle, USA, August 2008.

[2] Norbert Egi, Adam Greenhalgh, Mark Handley, Mickael Hoerdt, Felipe Huici, and Laurent Mathy. Towards high performance virtual routers on commodity hardware. In *Proceedings of ACM CoNEXT 2008*, Madrid, Spain, December 2008.

[3] Mark Handley, Eddie Kohler, Atanu Ghosh, Orion Hodson, and Pavlin Radoslavov. Designing extensible ip router software. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 14–14, Berkeley, CA, USA, 2005. USENIX Association.

[4] Eddie Kohler, Robert Morris, Benjie Chen, John Jahnotti, and M. Frans Kasshoek. The click modular router. *ACM Transaction on Computer Systems*, 18(3):263–297, 2000.

[5] 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 SIGCOMM'08*, Seattle, USA, August 2008.