

A Platform for High Performance and Flexible Virtual Routers on Commodity Hardware

Norbert Egi[§] Adam Greenhalgh[‡] Mickaël Hoerd[§] Felipe Huici^{*}
Panagiotis Papadimitriou[§] Mark Handley[‡] Laurent Mathy[§]

[§]Computing Dept., Lancaster University, UK

{*n.egi, m.hoerd, p.papadimitriou, l.mathy*}@lancaster.ac.uk

[‡]Dept. of Computer Science, University College London, UK

{*a.greenhalgh, m.handley*}@cs.ucl.ac.uk

^{*}NEC Europe, Heidelberg, Germany

felipe.huici@nw.neclab.eu

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 modern PC hardware. We further discuss our design choices in order to achieve both high performance and flexibility for packet processing.

Categories and Subject Descriptors

C.2.6 [Computer Communication Networks]: [Internetworking-Routers]

General Terms

Design

Keywords

Virtualization, Routers, Commodity Hardware

1. INTRODUCTION

Recent research has shown that modern PCs are a viable platform for the implementation of high-performance software routers [1, 2]. The functionality provided by software routers combined with recent virtualization technologies allows multiple instances of routers to run concurrently on a single box while offering highly configurable forwarding planes and custom routing protocols. Some recent PC-based virtual router prototypes [3, 4] have been proposed, but none of them exploit recent advances in commodity hardware such as multi-core CPUs or network interface cards with hardware multi-queuing.

In [1] we showed that the main performance bottleneck for PC-based software routers is main memory access. With the memory subsystem as the limiting factor for at least the next few years, the number of spare CPU cycles is likely to increase. To exploit these resources we can certainly consolidate a set of software routers running concurrently on the same hardware, inline with conventional server virtualization. However, the short-lived nature of packets inside a router makes router virtualization challenging and certainly more demanding. With performance and flexibility as primary goals,

the following question is raised: is it possible to have the best of two worlds, building a flexible virtual router platform that also reaches the performance limits of the underlying hardware?

Motivated by this challenge, we present a new platform for software virtual routers on commodity hardware. The platform leverages modern and emerging hardware trends to provide: (i) consolidation of virtual data planes onto a common forwarding domain, (ii) highly configurable forwarding planes for advanced programmability, and (iii) the forwarding tree¹ as the basic resource allocation unit for the mapping of virtual router components to cache hierarchies.

2. PLATFORM DESIGN OVERVIEW

The design of our virtual router platform is mainly driven by performance and flexibility. We use Xen's [5] paravirtualization to host the guest domains and Click [6] for packet processing and forwarding.

The platform comprises the following basic components (figure 1): (i) a management domain for the management of guest domains, (ii) an isolated driver domain (IDD) for aggregated packet forwarding, and (iii) a number of guest domains for hosting control planes (one per virtual router), and optionally forwarding planes when increased isolation and safety properties are required.

The IDD is a virtual machine that has the physical devices mapped to it and runs the appropriate device driver and router software required to host our forwarding planes (FPs). The IDD hosts the merged forwarding paths and provides the ability to control and configure the individual FPs to their respective guest domains. The merging process enables the consolidation of all FPs within the IDD, allowing a large number of virtual routers to share common network interfaces. In [1], we showed that forwarding within a common domain provides significantly higher performance than forwarding in the separate guest domains by avoiding costly per-packet hypervisor domain context switches. Fig. 1 depicts this configuration with FP1 running in the IDD.

The platform supports two additional packet forwarding configurations: (i) splitting a FP between an IDD and a separate guest domain while using local I/O channels for inter-domain communication (e.g., FP2 and FP2' in Fig. 1); and (ii) mapping interfaces directly into guest domains so that each FP resides in a separate guest domain (e.g., FP3 in Fig. 1). Configuration (i) can be used to

¹A forwarding tree is the set of packet processing elements necessary to move a packet from a single input to all possible outputs.

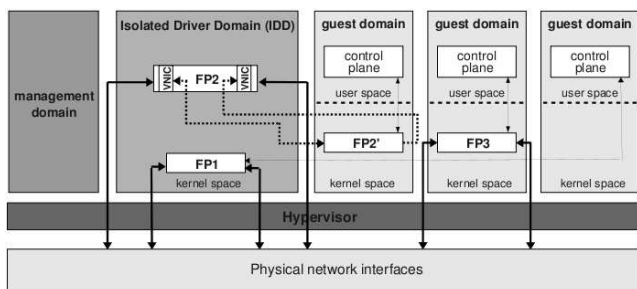


Figure 1: Platform overview.

safely run FPs that include untrusted Click elements without compromising the performance and safety of other virtual routers' FPs. In configuration (ii) we can directly access the network interfaces from a guest domain, but only by allocating whole (unshared) interfaces to the domain, resulting in coarse network resource sharing. To overcome this issue, we need virtualized network interface support both on the hardware side (e.g., VMDq) as well as on the software side (i.e., in the Xen Hypervisor).

The control planes for the virtual routers reside in the guest domains. The platform supports off-the-shelf control plane solutions running in user space, such as the *Extensible Open Router Platform (XORP)* or *Quagga*.

3. FORWARDING PATH ARCHITECTURE

In order to maximize the performance of PC-based virtual routers, the CPU cores and the cache memory hierarchies need to be carefully exploited. A cache hierarchy is the set of caches within one CPU package, with a PC potentially containing several CPUs and thus several hierarchies. As a result, the aim when implementing a virtual router should be to keep a packet as deep as possible inside a cache hierarchy (i.e., close to the cores) while distributing the packet processing over as many spare cores as possible within the same cache hierarchy. This ensures that processing is not CPU-limited since multiple cores are in use, while reducing expensive accesses to main memory.

A software router's internal organization can be viewed as a graph of interconnected packet processing elements. With this in mind, achieving high performance is non-trivial: while allocating a whole router onto a cache hierarchy is possible, it may not always be desirable, since confining the whole of a software router to only a subset of the cores in the system would result in poor resource utilization.

Even though it is not possible to know in advance to which output interfaces packets have to be switched, they should be kept within a single cache hierarchy whilst distributing the packet processing over as many spare cores as possible within that hierarchy. The key to solving this is to realize that a router's graph organization can be decomposed into a series of *forwarding trees*. Each of these forwarding trees is associated with an input interface and represents all of the possible forwarding paths followed by packets entering through that interface. Fig. 2 illustrates such a configuration where two forwarding trees are rooted at the input interfaces, each with its own set of independent elements. The advantage of using a forwarding tree is that its elements can all be allocated to the same cache hierarchy, thus confining packets to this hierarchy and reducing main memory accesses. In addition, forwarding trees compose a smaller allocation unit than routers, providing more flexibility when exploiting hardware resources and implementing fairness. In order to avoid conflicts among the trees during the lookup oper-

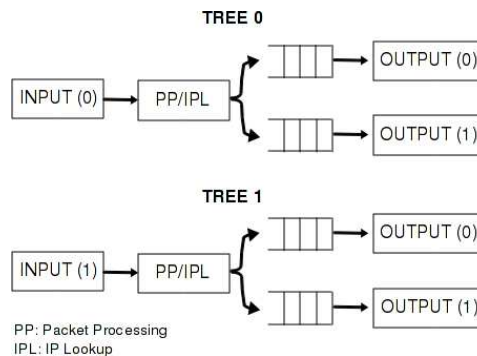


Figure 2: Tree-based IP forwarding.

ation, we replicate the forwarding information base on all cache hierarchies.

Although forwarding trees can be allocated to cache hierarchies independently from each other, they are not, in fact, completely independent from each other. Indeed, after the lookup operation, forwarding trees belonging to the same virtual router might have to send packets to the same output interfaces, and exclusive access to these interfaces must therefore be granted to each forwarding tree for transmission. A possible solution is to use a locking mechanism such as a Linux spinlock for each one of the output interfaces. However, our experiments show that spinlocks can cause cache misses, thus degrading performance.

This locking contention can be entirely removed if the outbound interface supports multiple hardware queues, at least one per cache hierarchy on the system but preferably one per CPU core. We are currently conducting experiments on a 10Gb/s network interface that supports multiple hardware queues and hardware classification in order to confirm this claim.

4. CONCLUSIONS

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

5. REFERENCES

- [1] N. Egi, A. Greenhalgh, M. Handley, M. Hoerd, F. Huici, and L. Mathy, "Towards high performance virtual routers on commodity hardware," in *Proceedings of ACM CoNEXT 2008*, Madrid, Spain, December 2008.
- [2] K. Argyraki, S. A. Baset, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, E. Kohler, M. Manesh, S. Nedveschi, and S. Ratnasamy, "Can software routers scale?" in *Proceedings of PRESTO '08*, Seattle, USA, August 2008.
- [3] S. Bhatia, M. Motiwala, W. Muhlbauer, Y. Mundada, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford, "Trellis: A platform for building flexible, fast virtual networks on commodity hardware," in *Proceedings of ACM ROADS '08*, Madrid, Spain, December 2008.
- [4] E. Keller and E. Grrn, "Virtualizing the data plane through source code merging," in *Proceedings of PRESTO '08*, Seattle, USA, August 2008.
- [5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *19th ACM Symposium on Operating Systems Principles*. ACM Press, October 2003.
- [6] E. Kohler, R. Morris, B. Chen, J. Jahnott, and M. F. Kasshoek, "The click modular router," *ACM Transaction on Computer Systems*, vol. 18, no. 3, pp. 263–297, 2000.