Extending Networking into the Virtualization Layer

Ben Pfaff* Justin Pettit* Teemu Koponen*
Keith Amidon* Martin Casado* Scott Shenker‡

ABSTRACT

The move to virtualization has created a new network access layer residing on hosts that connects the various VMs. Virtualized deployment environments impose requirements on networking for which traditional models are not well suited. They also provide advantages to the networking layer (such as software flexibility and well-defined end host events) that are not present in physical networks. To date, this new virtualization network layer has been largely built around standard Ethernet switching, but this technology neither satisfies these new requirements nor leverages the available advantages.

We present Open vSwitch, a network switch specifically built for virtual environments. Open vSwitch differs from traditional approaches in that it exports an external interface for fine-grained control of configuration state and forwarding behavior. We describe how Open vSwitch can be used to tackle problems such as isolation in joint-tenant environments, mobility across subnets, and distributing configuration and visibility across hosts.

1 Introduction

Virtualization has changed the way we do computing; for instance, many datacenters are entirely virtualized to provide quick provisioning, spill-over to the cloud, and improved availability during periods of disaster recovery. Further, the adoption of virtualization shows no signs of slowing. A recent Gartner report estimates that while 12% of x86 workloads are virtualized today, this number will grow to 61% by 2013. And recently, Intel has stated their goal for all end hosts to be virtualized [12].

With the proliferation of virtualization, a new network access layer is emerging that provides inter- and intra-VM connectivity and is evolving many of the same functions provided by the physical layer. Even today, this layer is providing connectivity to tens of VMs per physical server.[1]

While virtualization's impact on computing is well known, its implications for networking are far less explored. In particular, virtualization imposes requirements on network mobility, scaling, and isolation that are far beyond what is required in most physical deployments. Seamless handling of mobility is a necessity, since VMs can freely migrate between hosts and scaling limits are tested because datacenters can support hundreds of thousands of VMs. Strong isolation is required in joint-tenant environments where tenants share the same physical infrastructure.

While imposing more stringent requirements, virtualization also provides features making networking easier. For example, in virtualized environments, the virtualization layer can provide information about host arrivals and movements. Similarly, multicast membership can be inferred through introspection within the virtualization layer. The topology also becomes more tractable because networking at the virtualization layer is composed entirely of leaf nodes.

Thus, networking in a virtualized world presents its own set of challenges and opportunities. However, the typical model for internetworking in virtualized environments is standard L2 switch [4, 16] or IP router [3] functionality within the hypervisor or hardware management layer. This virtual networking component manages communication between co-located virtual machines, and connectivity to the physical NIC. There have been some attempts to adapt the virtual network layer to its unique set of properties, but none of the implementations adequately handle the full range of challenges.[2]

We believe this is in part due to the use of standard L2/L3 technologies in all the major virtual environments. In this paper, we present Open vSwitch, a new virtual switch that was purpose-built for use in virtualized environments. Open vSwitch differs from traditional approaches in that it exports an interface for fine-grained control of the forwarding, which can be used to support QoS, tunneling, and filtering rules. Open vSwitch also supports a remote interface that allows for the migration of configuration state (useful for attaching networking policies to VMs). In addition, the implementation of Open vSwitch has a flexible, table-based forwarding engine which can be used to logically partition the forwarding plane. Open vSwitch is compatible with most Linux-based virtualization environments including Xen, XenServer, KVM, and QEMU. Open vSwitch is open source, and available under the Apache license at http://openvswitch.org.

The rest of the paper is organized as follows. We describe how the virtual networking environment differs from the physical in §2, and present the design and use of

*Nicira Networks
‡UC Berkeley, Computer Science Division

[1] Running 40-60 VMs per a host is not rare. However, we know of a deployment running as many as 120 VMs per a host.

[2] VMware's virtual switch is the most advanced, using knowledge from the virtualization layer to affect forwarding.

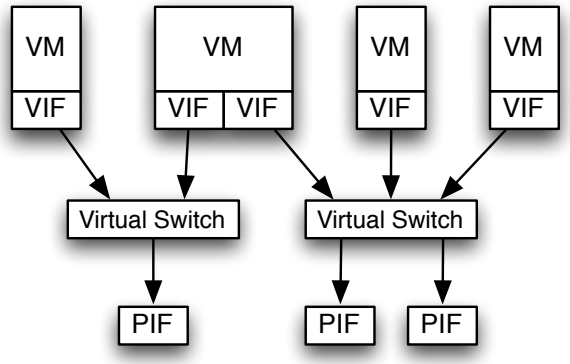

Figure 1: Virtual switches connect virtual interfaces (VIFs) to physical interfaces (PIFs).

Open vSwitch in §3. In §4 we present the implementation and analyze its performance. We discuss related work in §5 and conclude in §6 with a discussion about the potential impact of the virtualization to physical networks.

2 Network Virtualization

Networking in virtual environments is typically realized as a simple L2 switch within the hypervisor or the management domain (as depicted in Figure 1). Rather than providing Virtual Machines (VMs) with direct access to the NICs, VMs are often connected to virtual interfaces (VIFs). Virtual switches provide connectivity between these VIFs and the physical interfaces (PIFs), and also handle traffic between VIFs colocated on the same physical host. However, unlike physical switches connecting hosts to a network, virtual switches reside in the host and are typically written entirely in software.[3] This frees the implementation from the rigidity of hardware and enables sophisticated forwarding functions and far quicker design cycles.

The trend of VM consolidation on multi-core servers is steadily increasing the number of VMs managed by a single virtual switch. It is not uncommon for a single server to host 40 or more VMs. In order to provide some visibility into and control of this new networking layer, virtual switches are starting to support basic management interfaces such as SNMP, ERSPAN, and a CLI.

While the management interfaces and forwarding functions match those of physical switches, the deployment environment differs in ways that can be leveraged by properly designed virtual switches. In particular, tight integration with the virtualization software allows the virtual switch to take advantage of information that isn't easily derivable by inspecting network traffic alone. For example, the virtualization layer can inspect the VM to determine the MAC addresses of each of its VIFs, whether or not the VIF is in promiscuous mode, what IP addresses are allocated to a particular VIF, and for which multicast addresses a host is listening. This information can be used to reduce flooding of broadcast and multicast traffic without relying on imperfect, in-network solutions such as IGMP snooping.

VM introspection can also characterize the host, allowing more efficient processing within the switch. While not a primary focus of this paper, this has implications for in-network packet processing such as deep-packet-inspection (DPI). For example, a DPI-enabled virtual switch could limit the DPI signatures to those that apply to the installed OS on the VM and correctly determine how to reassemble fragments to match the guest.

Unlike physical environments in which inferring host events is notoriously difficult, in virtualized settings the hypervisor or hardware management layer orchestrates VMs by powering them on and off, changing their network connectivity parameters, and migrating them from one host to another. This additional information can be leveraged by the virtual networking layer to better manage configuration and forwarding state. For example, a virtual switch can set up MAC forwarding entries for all active VMs, and can limit other configuration state (*e.g.*, ACLs) to those associated with the currently attached VMs. For a traditional network switch, there are only indirect means to learn about these events, but for the network virtualization layer this information is available via communication with the non-networking parts of the host virtualization software. Reliable host events are used for distributed virtual switch implementations which are just starting to gain popularity [7, 20].

In addition, virtualized environments have characteristics that can simplify the implementation of the network virtualization layer. In most datacenter and cloud deployments, the network virtualization layer—which is effectively a software-based implementation of a switch or a router for VMs—operates like a leaf of the physical network topology. Its position as leaf removes the burden of participating in any network routing protocols; for example, a virtual switch need not run spanning tree.

Virtualized environments also pose new service requirements, particularly for multi-tenant datacenters. For cloud providers (like Amazon) to most efficiently utilize hardware resources while preserving security, multiple clients should share the same physical infrastructure but be logically isolated despite VM migration between hosts. In addition, some cloud providers would like to offer broadcast domains to each of their clients, requiring dynamic virtual overlays at the L2 level. Moreover, VMs from the same client can be distributed across multiple physical hosts. Depending on the offered service model, this could require support for distributed

[3] As discussed in §4, the fast path of the virtual networking layer may eventually be implemented in hardware.

QoS and policing. Finally, VM migration techniques currently enable movement only within the same subnet, yet often the hosting facilities are large, consisting of thousands of physical servers. It is also desirable to partition control interfaces, and resources to be available and accountable per-tenant.

None of these challenges are new; indeed, the literature is full of proposals to solve them. Therefore, we emphasize Open vSwitch is not intended to invent new solutions, but rather to establish a switch platform that can take advantage of the virtual environment and is flexible enough to implement the various approaches necessary to satisfy the full array of requirements, all without incurring significant performance overheads as compared to existing virtual switches.

3 Open vSwitch

In this section we describe Open vSwitch and how to use it, leaving implementation details for later.

3.1 Design Overview

Open vSwitch is a software switch that resides within the hypervisor or management domain (*e.g.*, Dom0 in Xen) and provides connectivity between the virtual machines and the physical interfaces. It implements standard Ethernet switching with VLAN, RSPAN, and basic ACL support. In a standalone configuration, it operates much like a basic L2 switch. However, to support integration into virtual environments, and to allow (logical) switch distribution, Open vSwitch exports interfaces for manipulating the forwarding state and managing configuration state at runtime. We describe these interfaces next.

Configuration. Through the configuration interface, a remote process can read and write configuration state (as key/value pairs), and set up triggers to receive asynchronous events about configuration state changes. While the functionality exposed over the interface will grow over time, it presently can turn on port mirroring (SPAN and RSPAN), apply QoS policies to interfaces, enable NetFlow logging for a VM, and bond interfaces for improved performance and availability.

In addition to remote configuration management actions, this interface provides bindings between network ports and the larger virtual environment. For example, the interface exposes the globally unique identifiers (UUIDs) for VIFs bound to the switch. As we discuss below, this information is essential for topology independent configuration declaration (in a manner similar to [21]).

Forwarding path. In addition to providing interfaces for managing configuration state, which are common in physical switches (*e.g.*, [10]), Open vSwitch provides a method to remotely manipulate the forwarding path. This allows an external process to write to the forwarding table directly, specifying how packets are handled based on their L2, L3, and L4 headers. The lookup can decide to forward the packet out of one or more ports, to drop the packet, or to en/decapsulate the packet. The forwarding path interface implements a superset of the OpenFlow [17] protocol.

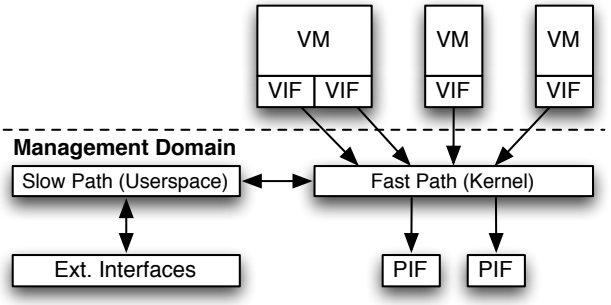

Figure 2: Architecture of the Open vSwitch

Connectivity management. Open vSwitch provides a local management interface through which the virtualization layer can manipulate its topological configuration. This includes creating switches (multiple virtual switches can reside on a physical host), managing VIF connectivity (for each connected VIF, a logical port is added to the switch), and managing PIF connectivity.

Underlying Open vSwitch is a flow-table forwarding model similar to that used by OpenFlow [17] and discussed more generally in [6]. The rationale for rule-based forwarding is that it enables near-arbitrary logical partitioning of the forwarding functions (this same method is used for arbitrary network partitioning schemes, see [19]). More specifically, it allows network configuration state and forwarding functions to be associated with a subset of the traffic whether from a single VIF, a single VM, or a group of VMs.

In its simplest deployments, Open vSwitch is much like a traditional physical switch within the virtualization layer. Each instance is separately administered through the management interfaces, providing visibility and control over inter-VM communication which is invisible to the first hop physical switch. However, the inclusion of interfaces for globally managing configuration and forwarding state enables distribution of the switch functions across multiple servers—effectively decoupling the logical network topology from the physical one. For example, a remote process, if integrated with the virtualization control platform, can migrate network configuration state with VMs as they move between physical servers. Most standard virtualization environments (*e.g.*, VMware, XenServer, and libvirt) provide hooks for such integration.

In addition, the ability to manipulate the forwarding table through an external interface allows low-level flow

state to migrate with a virtual machine. For example, this could be used to migrate existing flow counters and ACLs. It also allows the migration of tunneling rules which can be used to support seamless migration between different IP subnets. We discuss this in more detail below.

3.2 Using Open vSwitch

We now describe common use cases for Open vSwitch in virtualized environments.

Centralized management. The interfaces for configuration management and asynchronous notification can be used to create a single logical switch image across multiple Open vSwitches running on separate physical servers (similar to [7, 20]). In essence, a global management process synthesizes a logical view of the switches and their configuration declarations and lets admins operate on that view instead of individual switches. Therefore, as VMs join, leave, and migrate, it is the responsibility of this management process to ensure any configuration state remains coupled to the logical entities for which it was declared.

We have used this interface to build a full CLI for configuring the network as a whole. The CLI presents the abstraction of a virtual port to which network configuration state can be applied (*e.g.*, VLANs, RSPAN, QoS policies, or ACLs). Each virtual port then corresponds to a unique VM and follows the VM throughout the network. As a result, it is possible to query and configure a collection of virtual switches as if they were a single switch.

Virtual Private Networks. A growing segment of virtualization is cloud hosting in which a third party hosts the virtual machine of multiple clients (tenants). Ideally, to best utilize the hardware, these joint-tenant environments should co-locate tenants on the same physical infrastructure while providing strong isolation guarantees.

In the same way that a group of physical machines can be connected to each other over a dedicated network, in a virtualized environment a collection of VMs can be connected to each other over a private, virtual network implemented on top of a shared physical network infrastructure.

If all of the VMs for a single tenant are on the same physical host, or if they are on separate hosts that each dedicate a NIC to connecting to an isolated physical switch, supporting these virtual private networks is simple. However, when VMs sharing a private network are spread across multiple hosts and/or multiple physical switches, the virtualization networking layer must support dynamic overlay creation.

Open vSwitch supports both VLANs and GRE tunnels [11]. While VLAN support is sufficient for small-scale deployments (each tenant is connected to a separate VLAN), the support for GRE is essential in overcoming the limitations VLANs face in larger deployments: the number of available VLANs is limited (compared to the size of many virtual environments), VLANs do not nest well, and, as a link layer concept, VLANs do not extend beyond LANs.

For Open vSwitch, GRE tunnels are a way to encapsulate an Ethernet frame inside an IP datagram to be routed from one subnet to another. As a global management process keeps track of the network locations of all of the VMs (on each private network), it can select the best way to forward packets from one VM to another while modifying flow tables accordingly in Open vSwitches: if the communicating VMs are on a single host, the virtual private network exists within a single Open vSwitch; if the VMs are all on the same subnet, the management process will use VLANs to establish the private network, and if the VMs span multiple subnets, it can use GRE tunnels.

Mobility between IP subnets. A well known limitation of today's commercial virtualization platforms is that migration must happen within a single IP subnet. This is due to the inability to maintain transport sessions over changes in endpoint address. Migration between subnets is desirable for a number of reasons. For example, single L2 domains have scalability limits requiring operators to segment their networks, thereby imposing artificial limits on mobility.

There are multiple ways to accomplish this with Open vSwitch and a global management process. The most straightforward is to use a model similar to Mobile IP in which a base Open vSwitch receives all packets for a given VM and then forwards the packet to the true location using tunneling. The global management process must manage the tunneling rules, keeping them consistent with VM locations in the network.

4 Implementation and Analysis

In this section we discuss the implementation of Open vSwitch and compare its performance to the Linux bridging code, which is the de-facto standard open source component for virtual networking today.

4.1 Implementation

The Open vSwitch implementation consists of two components: a kernel-resident "fast path" and a userspace "slow path". The fast path implements the forwarding engine which is responsible for per-packet lookup, modification and forwarding. It also maintains counters for each forwarding table entry.

The majority of the functionality is implemented within the slow path, which is intended to run within the VM management domain. The slow path implements the

forwarding logic, including MAC learning and load-balancing over bonded interfaces. It also implements the remote visibility and configuration interfaces, such as NetFlow, OpenFlow, and remote management protocols.

Keeping the fast path simple (it has 3000 lines of code compared with over 30,000 lines for the slow path) has two notable benefits. First, because it is the speed critical portion of the system, it should be implemented within the kernel and therefore is necessarily system-specific. This limits the amount of code that has to be written during porting. Earlier versions of Open vSwitch have been ported to multiple non-Linux platforms. Second, keeping the datapath forwarding model simple (as a flow lookup) eases the migration path to hardware accelerated forwarding which is already gaining momentum in mass-produced NICs. We discuss this trend below.

While it has been ported to other OSes, Linux-based virtualization platforms are its primary environments. To simplify integration with them, Open vSwitch emulates the interfaces of VDE [9] and the Linux bridging code, both of which are commonly used in virtual deployments. As a result, Open vSwitch can be used as a drop-in replacement for the virtual switches used by Xen, XenServer, and KVM. In fact, most of the existing tools and utilities can be run unmodified against Open vSwitch due to its exporting of ioctl, sysfs, and proc filesystem interfaces identical to these legacy virtual switch implementations.

4.2 Performance

We now compare the performance of Open vSwitch against that of the Ethernet bridge implementation in the Linux kernel. The Linux Ethernet bridge, which has been used and maintained since before the release of Linux 2.4 in 2002, is an entirely in-kernel implementation. Each packet received by the Linux bridge is forwarded based solely on its destination MAC, using a MAC learning table that is updated dynamically based on each packet's source MAC.

Figure 3 compares the effective throughput that Open vSwitch achieves for various flow sizes against the Linux bridge. The measurements were executed using NetPIPE 3.6.2 inside Debian GNU/Linux 4.0 VMs within Citrix XenServer 5.5.0. The VMs were connected with a crossover cable with a 1 Gbps NIC at each end. The graph confirms that Open vSwitch performance is comparable to that of the Linux bridge. (The anomalies in the performance of the Linux bridge at transfer sizes of 192 kB and 4 MB remained after repeated re-testing.)

This simple transfer scenario, in which the majority of packets can be handled by the fast path, is the best case for Open vSwitch. It is similar to the common case encountered in early Open vSwitch deployments, in which packets are rarely passed up to the slow path. The worst case scenario for Open vSwitch, in which every arriving packet must enter the slow path, can saturate the limited kernel/userspace channel and cause significant packet loss. However, this situation is very unlikely in practice with legitimate traffic. See [6] for more data on the speed of flow table-based forwarding.

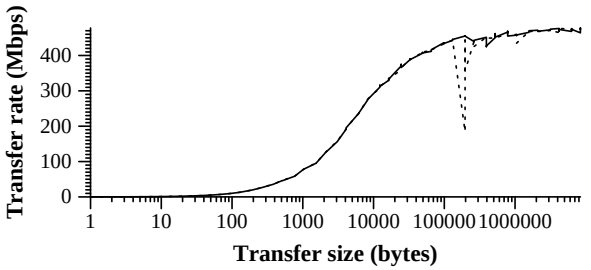

Figure 3: Throughput versus flow size for Xen virtual machines with Linux bridge (dashed) and Open vSwitch (solid).

4.3 Hardware Accelerated Forwarding

As the number of cores per server increases, so does the number of virtual machines that can be hosted on a server and their aggregate bandwidth requirement. Today it isn't uncommon for a server to host 40 or more VMs. Unsurprisingly, this trend drives new hardware technologies to accelerate forwarding in virtualized environments. For example, Intel's VMDQ [15] provides hardware accelerated switching in the NIC as do numerous SR-IOV [14] NICs.

As these technologies proliferate, we expect the fast path to drop into hardware, and for the slow path to instruct the network silicon to transmit (and receive) packets directly into (and from) the corresponding VM.

We note another approach gaining traction is to forgo processing on the end-host entirely and instead rely on the first hop switch to handle inter- and intra-VM traffic (*e.g.*, VNTag [18] and VEPA [8]). This requires all traffic to transit the first hop link rather than taking advantage of on-host switching bandwidth for inter-VM traffic. However, even in this case, the Open vSwitch control layer can be used to manage the switch configuration state if the appropriate interfaces exist on the first hop switch.

5 Related Work

All useful virtualization platforms have some form of networking support. For example, VMware's recently released distributed switch [20], and Linux-based virtualization environments generally use the bridging code or the Virtual Distributed Ethernet (VDE) switch [9]. Networking vendors are also starting to create virtual switches [7] and other virtual network devices [1] for the virtualization layer.

In most cases, these approaches use standard L2/L3 forwarding and standard (or proprietary) interfaces for management, and have not demonstrated the flexibility needed to overcome some of the challenges discussed in this paper.

There has also been a lot of recent work focused on virtualizing the network infrastructure itself (*e.g.*, [2, 3, 5, 13, 21]). In general, this line of work seeks to gain the advantages of virtualization, such as isolation, flexibility, and mobility, *within* the switches and routers constituting the network core. In contrast, our work focuses on the emerging networking layer *at the end host* and on exploring how the differences in deployment environment affect the design of the networking technologies which are deployed within them.

6 Discussion

So far, we have focused on the design and features of Open vSwitch and how it compares to existing approaches, but its broader implications should not be overlooked. There has been a growing tension between the need to keep networks simple (so we can understand and operate them) and the need to meet the many networking requirements of modern enterprises and datacenters. Recently, simplicity has been losing out to functionality, and today's enterprise and datacenter networks require large trained staffs to operate.

Virtualization has an opportunity to change this. In virtualized environments, the network extends to the host, where Open vSwitch resides. And if the environment is *completely* virtualized, then *every* leaf node is a virtual switch. This allows us to make a clean distinction between the edge and core pieces of the network infrastructure. Note that the features discussed in §3 only required the cooperation of the virtual (edge) switches, and did not rely on any coordination with the physical (core) switches. By themselves, the virtual switches can implement security policies, virtual private networks, visibility, and VM mobility. This allows the core infrastructure to remain very simple, focused only on providing connectivity between relatively static hosts.

Thus, the longstanding tension between network simplicity and functionality has been somewhat resolved. The virtual switches, which are far more flexible and integrate more tightly with host software than do physical switches, can implement all of the sophisticated functionality in a straightforward manner. The core network infrastructure remains blissfully ignorant of these advanced features, and need only support simple connectivity between hosts.

The advent of virtualization made virtual switches a necessity. What we've shown here is that for a wide array of advanced functionality, virtual switches aren't just necessary, they are sufficient. Thus, the advent of sophisticated virtual switches like Open vSwitch should be seen as an opportunity to return to the days of simple networking hardware.

7 Acknowledgements

Peter Balland, Natasha Gude, and Daniel Wendlandt helped formulate the Open vSwitch control requirements. Jean Tourrilhes at HP Labs also provided input. Simon Crosby and Ian Campbell at Citrix provided invaluable guidance for integrating into virtual environments. We also thank the anonymous reviewers.

8 References

[1] Altor Networks. `http://altornetworks.com`, July 2009.

[2] T. Anderson, L. Peterson, S. Shenker, and J. Turner. Overcoming the Internet Impasse through Virtualization. *Computer*, 38(4):34–41, 2005.

[3] F. Anhalt and P. V.-B. Primet. Analysis and Evaluation of a Xen Based Virtual Router. Technical Report 6658, Inria, Sept. 2008.

[4] 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 *Proc. of SOSP*, Oct. 2003.

[5] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In VINI Veritas: Realistic and Controlled Network Experimentation. In *Proc. of SIGCOMM*, 2006.

[6] M. Casado, T. Koponen, D. Moon, and S. Shenker. Rethinking Packet Forwarding Hardware. In *Proc. of HotNets*, Nov. 2008.

[7] Cisco. Nexus 1000V Series Switches. `http://www.cisco.com/en/US/products/ps9902`, July 2009.

[8] P. Congdon. Virtual Ethernet Port Aggregator Standards Body Discussion. `http://www.ieee802.org/1/files/public/docs2008/new-congdon-vepa-1108-v01.pdf`, Nov. 2008.

[9] R. Davoli. VDE: Virtual Distributed Ethernet. In *Proc. of TRIDENTCOM*, Feb. 2005.

[10] R. Enns. NETCONF Configuration Protocol. RFC 4741, IETF, Dec. 2006.

[11] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. Generic Routing Encapsulation (GRE). RFC 2784, IETF, Mar. 2000.

[12] P. Gelsinger. Keynote at Citrix Synergy. May 2009.

[13] A. Greenhalgh, F. Huici, M. Hoerdt, P. Papadimitriou, M. Handley, and L. Mathy. Flow Processing and The Rise of Commodity Network Hardware. *SIGCOMM CCR*, Apr. 2009.

[14] Intel. Single Root I/O Virtualization and Sharing 1.0 Specification. `http://www.pcisig.com/members/downloads/specifications/iov/sr-iov1.0_11Sep07.pdf`, Sept. 2007.

[15] Intel. Virtual Machine Device Queues White Paper. `http://software.intel.com/file/1919`, 2007.

[16] R. Malekzadeh. VMware for Linux Networking Support. `http://web.archive.org/web/19991117164603/vmware.com/support/networking.html`, 1999.

[17] OpenFlow Consortium. OpenFlow Specification 0.9.0. `http://www.openflowswitch.org`, July 2009.

[18] J. Pelissier. VNTag 101. `http://www.ieee802.org/1/files/public/docs2009/new-pelissier-vntag-seminar-0508.pdf`, May 2008.

[19] R. Sherwood, M. Chan, G. Gibb, N. Handigol, T.-Y. Huang, P. Kazemian, M. Kobayashi, D. Underhill, K.-K. Yap, and N. McKeown. Carving Research Slices Out of Your Production Networks with OpenFlow (SIGCOMM Demo), Aug. 2009.

[20] VMware. vNetwork Distributed Switch. `http://www.vmware.com/products/vnetwork-distributed-switch`, July 2009.

[21] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford. Virtual Routers on The Move: Live Router Migration as A Network-management Primitive. In *Proc. of SIGCOMM*, Aug. 2008.