sNICh: Efficient Last Hop Networking in the Data Center

Kaushik Kumar Ram, Jayaram Mudigonda, Alan L. Cox, Scott Rixner, Partha Ranganathan, Jose Renato Santos

HP Laboratories
HPL-2010-154

Keyword(s):

No keywords available.

Abstract:

Virtualization has fundamentally changed the data center network. The last hop of the network is no longer handled by a physical network switch, but rather is typically performed in software inside the server to switch among virtual machines hosted by that server. In this paper, we present the concept of a sNICh, which is a combination of a network interface card and switching accelerator for modern virtualized servers. The sNICh architecture exploits its proximity to the server by separating the network switching tasks between hardware and software efficiently. This enables the sNICh to address the resource intensiveness of software virtualization and the scalability limits of current hardware support. The sNICh utilizes a flow-based approach, in which hardware performs basic switching while software handles flow setup based on packet filtering rules. The sNICh minimizes I/O bandwidth utilization by transferring, whenever possible, inter-virtual machine traffic within the main memory. We also present a preliminary evaluation of this architecture using software emulation. We compare the performance of the sNICh with two existing software solutions in Xen, the Linux bridge and Open vSwitch. Our results show that the sNICh out-performs both these existing solutions and also exhibits better scalability.

External Posting Date: October 21, 2010 [Fulltext]   Approved for External Publication
Internal Posting Date: October 21, 2010 [Fulltext]
To be presented at ANCS 2010: The 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, San Diego, CA, Oct 25-26, 2010.

sNICh: Efficient Last Hop Networking in the Data Center

Kaushik Kumar Ram
Rice University
kaushik@rice.edu

Jayaram Mudigonda
HP Labs
jayaram.mudigonda@hp.com

Alan L. Cox
Rice University
alc@rice.edu

Scott Rixner
Rice University
rixner@rice.edu

Partha Ranganathan
HP Labs
Partha.Ranganathan@hp.com

Jose Renato Santos
HP Labs
joserenato.santos@hp.com

ABSTRACT

Virtualization has fundamentally changed the data center network. The last hop of the network is no longer handled by a physical network switch, but rather is typically performed in software inside the server to switch among virtual machines hosted by that server.

In this paper, we present the concept of a *sNICh*, which is a combination of a network interface card and switching accelerator for modern virtualized servers. The sNICh architecture exploits its proximity to the server by separating the network switching tasks between hardware and software efficiently. This enables the sNICh to address the resource intensiveness of software virtualization and the scalability limits of current hardware support. The sNICh utilizes a flow-based approach, in which hardware performs basic switching while software handles flow setup based on packet filtering rules. The sNICh minimizes I/O bandwidth utilization by transferring, whenever possible, inter-virtual machine traffic within the main memory.

We also present a preliminary evaluation of this architecture using software emulation. We compare the performance of the sNICh with two existing software solutions in Xen, the Linux bridge and Open vSwitch. Our results show that the sNICh out-performs both these existing solutions and also exhibits better scalability.

1. INTRODUCTION

Virtualization has become an integral component of the modern data center. Virtualization technologies have advanced to simplify server allocation, failover, and consolidation. However, networking technologies have not kept pace with this relatively new use of virtualization in the data center. Several disparate networks remain in the data center, and network performance and functionality can vary across the data center. Fortunately, the rapid growth in network link bandwidth provides the opportunity for network consolidation, whereby all traffic in the data center would be carried over a single network. While advances in network quality of service can facilitate such convergence by isolating traffic, virtualization dramatically complicates such network consolidation. Ultimately, the communication endpoints within the data center are virtual machines (VMs), not physical servers. Therefore, successful networking in the data center requires network features for protection, isolation, and allocation to operate all the way to the VM.

While data center switches provide these capabilities at high performance, there is still the issue of the last hop switch. In most data centers, the last hop switch operates in software either within the hypervisor or a dedicated driver domain. The Linux Ethernet bridge is used for this purpose in Xen. Cisco and VMware have made this last-hop switch look and behave like other switches in the data center [32]. However, there are significant software overheads inherent in data center switching that make this an inefficient solution. Cisco and VMware have also developed an alternative solution, in which an external switch is used as the last hop switch [25]. This entails routing all traffic, even traffic among VMs co-located on the same physical server, to the external switch. This inherently wastes I/O bandwidth within the server and network link bandwidth between the server and the switch.

These existing solutions for the last hop switch represent two ends of the spectrum: switching entirely in software within the server and switching in hardware outside the server. A middle ground in the design space is to switch packets within the server's network interface cards (NICs). In fact there exist direct-access NICs which provide this functionality [23]. The primary benefit of a direct-access NIC is that it entirely bypasses any software intermediary. So the VMs can directly send and receive network packets to/from the NICs. But this solution is not widely used since these NICs only implement rudimentary switching functionality.

This paper presents the sNICh architecture to efficiently support the last hop switch in the data center. The sNICh is a combined network interface card (NIC) and data center switching accelerator. But unlike existing solutions, the sNICh supports all data center switching functionalities. This enables protection, isolation, and allocation to be performed uniformly across the data center, in spite of VM placement and migration.

The sNICh architecture exploits its proximity to the server by separating the network switching tasks between hardware and software efficiently. This simplifies the delivery of complex switching functionality needed in the data center. For example, while basic packet switching is implemented in hardware, packet filtering using access control lists (ACLs) is performed in software. But this separation by itself is not useful if every packet has to traverse the software path. Instead, the packets are switched on a per-flow basis. The flows are validated once in software and then the validated flows are cached in hardware. Thus subsequent packets belonging to the validated flows are completely handled in hardware itself. This enables the sNICh architecture to address the resource intensiveness of software virtualization and the scalability limits of current hardware support.

Further, since the sNICh is aware of all the VMs on the server, it can manage the network traffic on a per-VM basis. The sNICh also extends network QoS all the way to the VMs, and thus addresses the problem of ensuring end-to-end QoS. Finally, the sNICh is architected to minimize the I/O bus utilization by transferring, wherever possible, all the inter-VM traffic within the main memory.

This paper presents a detailed description of the sNICh architecture. In particular, it explains how packet switching, packet filtering, and packet copying are performed in this architecture. Packet switching is implemented using the flow-based approach in hardware. Packet filtering using ACLs is performed primarily in software but flow entries based on matching ACL rules are cached in hardware. Packet copying is offloaded to DMA engines to avoid wastage of I/O bandwidth. This paper also presents a preliminary performance evaluation of the sNICh architecture. The evaluation is based on a sNICh prototype where the sNICh hardware is emulated in software. Specifically, the performance of the sNICh is compared to two software switching solutions in Xen, namely the Linux Ethernet bridge and Open vSwitch. The results show that the sNICh out-performs both these existing solutions and also exhibits better scalability than them.

The rest of this paper is organized as follows: in Section 2 we provide an overview of data center networking and discuss the current approaches to last-hop switching to virtual machines. Section 3 describes the sNICh architecture and how it takes advantage of its tight integration with the server. Section 5 presents a performance evaluation of the sNICh architecture. Section 6 reviews related work and finally Section 7 summarizes our conclusions.

2. BACKGROUND

The data center is becoming one of the most critical components of the modern computing infrastructure. This trend has manifested in several ways. Primarily, data intensive applications, such as Google's search engine, can only operate in large scale data centers. However, even smaller applications—workplace applications, such as document editors and spreadsheets, are migrating to the data center. Further, the utility computing model is emerging, whereby it is cost efficient to "rent" time in a large scale data center, enabling clients to quickly scale up or down the amount of computing resources at their disposal.

2.1 Data Center Networking Challenges

To efficiently serve this ever increasing number and diversity of applications and customers, data centers must address two inefficiencies: server sprawl and multiple poorly utilized networks.

Physical servers are rarely shared across multiple clients, and in many cases not even across application instances of the same client, so that the necessary performance SLAs and the inter-customer isolation can be achieved. Typically these servers are under-utilized and are wasteful of power [12].

Most data centers also contain several parallel networks: a traditional Ethernet; a Fibre Channel network for storage traffic; and an InfiniBand fabric to support cluster traffic. These parallel networks are not cost-effective for several reasons; they cost more to build, require multiple administrators, complicate cabling, waste rack space and energy.

Virtualization offers a promising avenue towards reducing server sprawl, particularly when combined with many core processors. Modern virtualization systems allow several servers to be effectively consolidated onto a single physical machine. Similarly, advances in Ethernet networking offer a promising avenue towards increasing network utilization in the data center. The rapid rise of Ethernet network link bandwidths combined with the advent of sophisticated switch-based mechanisms—such as VLANs, ACLs, and link schedulers— for safely multiplexing different clients and traffic types can facilitate fabric consolidation.

The networking subsystems of virtualized servers, however, present a major impediment for both server and fabric consolidation. Data center networks and the server I/O subsystems are both architected in a way that expects the physical server to be an *end-point* in the network, and do not efficiently support a virtualized server, which is in reality a *network* in itself of VMs. Lack of efficient support for switching and for mechanisms that aid in ensuring isolation (such as ACLs, VLANs and QoS) causes the following two major problems in data center networks:

1. Lack of efficient switching support within the server can affect server densities in the near-future for two reasons. First, in most modern data centers, inter-server communication is already significant, and is expected to increase further [9]. For instance, in Amazon's EC2 utility data center, a request from an external client machine can make as many as 100 different servers exchange messages among them [5]. Second, the increasing core counts on processor chips can easily be utilized by co-locating the servers of such applications on the same physical machine. However, this will not be possible if the networking subsystem cannot keep up to provide efficient inter-VM packet switching and isolation.

2. Lack of efficient access control within a server complicates fabric consolidation. The central problem in fabric consolidation is to isolate different clients and traffic types from hurting each other when forced to share a common switch or link. For instance, a buggy (or malicious) client should not be allowed to direct its storage traffic to another client's parallel program VMs, causing serious drops in their synchronization traffic. To fully ensure such isolation, one must enforce the access restrictions (and QoS guarantees) on *all* hops of an end-to-end path. However, in virtualized servers, the real end-points are the VMs, and hence the end-to-end path extends through the server, involving the I/O subsystem as the last-hop. If the I/O subsystem does not extend the isolation and allows traffic from different clients and different types to interact, it renders the isolation enforced in the greater data center network completely useless and makes fabric consolidation impossible.

2.2 Network Virtualization

Current state of the art network subsystem architectures for data center servers can be classified into three main categories.

The first category of systems (shown in Figure 1) includes a simple NIC that is virtualized by a software intermediary (such as the driver domain of Xen). These implementations (because they are in software) tend to have a rich set of packet processing functions such as ACL matching and link-scheduling. However, in most cases they cannot sustain high throughput for three reasons. First, the cost of supporting advanced switching functionalities like packet filtering to enforce access control and QoS can be expensive in software. Second, regardless of how expensive the packet processing itself is, merely getting the packet to and from the software intermediary can be very resource intensive [27]. Third, parallelizing these software implementations to take advantage of multiple processor cores remain challenging;

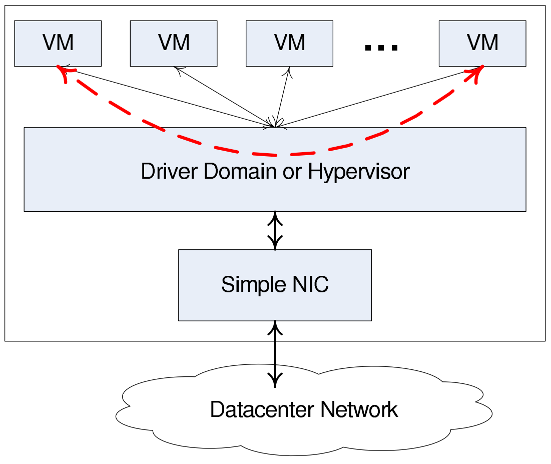

Figure 1: ***State of the Art in virtualized I/O subsystems–The first type. These architectures mostly rely on software–either the hypervisor or a privileged domain such as a driver domain—to virtualize a simple standard NIC. As the dotted arrow shows, the packet switching happens entirely in the software intermediary.***

it has been shown that even a judicious mapping of multiple driver domain threads to cores can often result in a net throughput *loss* [33].

The second category of systems (shown in Figure 2) employ more sophisticated NICs with multiple *contexts* that present virtual NIC (vNIC) interfaces to individual VMs [34, 24]. A VM that has a vNIC allocated to it can completely bypass the software intermediary and access the NIC directly. However, today most of these NICs only implement a rudimentary form of switch which does not support any advanced switching features. Further, these NICs waste substantial I/O bandwidth because they always transfer the full packet payload for each inter-VM packet twice over the I/O bus (to and from the NIC).

The third approach tries to leverage the functionalities which already exist in today's data center switches. This approach—being promoted by the industrial alliance between Cisco and VMWare—uses an external switch for switching *all* packets including the inter-VM traffic (as shown in Figure 3) [25]. In this architecture, a server agent and the external switch attach a special label to each packet that identifies the VM the packet belongs to. While the server agent uses this label to demultiplex the packets into the per-VM receive queues, the external switch uses it to enforce per-VM access controls and QoS. This also simplifies management, since *all* traffic from within the server now transits a traditional switch and hence can be managed by a network manager system.

Today, there are not many systems of this kind avail-

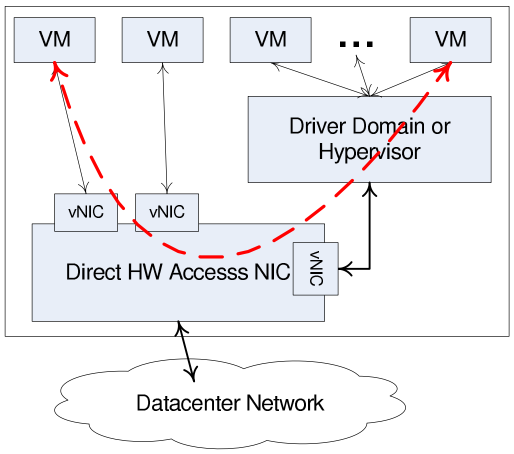

Figure 2: ***State of the Art in virtualized I/O subsystems–The second type. These architectures employ more sophisticated NICs that allow a subset of the VMs to directly access the hardware and support rudimentary switching.***

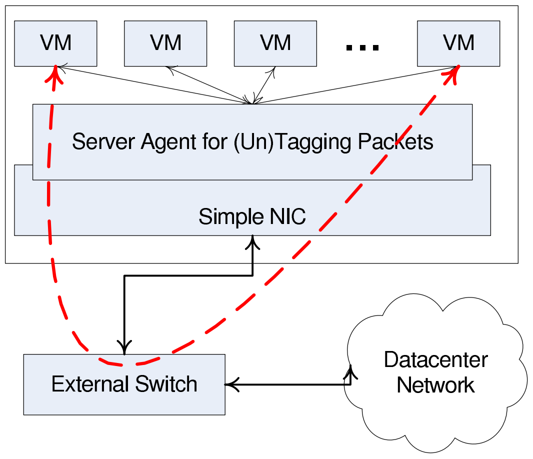

Figure 3: ***State of the Art in virtualized I/O subsystems–The third type. The servers blindly forward* all *packets to the external switch which then manages the traffic on a per-VM basis to ensure isolation and QoS guarantees.***

able for experimentation, however, a server agent implemented in software very likely incurs a good fraction of the CPU overhead of the software-based virtualization systems discussed above. Further, similar to the direct hardware access systems, this approach can result in substantial wastage of network bandwidth in addition to the I/O bandwidth, since all the packets from inter-VM traffic always travel all the way to the external switch.

To summarize, modern servers are networks themselves, yet they are not effectively integrated into the data center network as a whole. New solutions are needed that provide sophisticated data center networking functionality more efficiently, while tightly integrating into the consolidated data center network. The sNICh provides an effective solution to these problems by integrating the capabilities of a data center switch into the network interface hardware of virtualized servers.

3. SNICH ARCHITECTURE

The sNICh, as the name suggests, is a combination of a NIC and a switch. The sNICh acts as both an interface between the server and the external network and as a switch for the virtual machines within the server. Figure 4 illustrates sNICh's high-level architecture, and its relationship with rest of the server and the greater data center network.

Getting a packet to and from a software intermediary can be expensive. Hence, the sNICh presents a regular NIC-like interface to multiple virtual machines. These interfaces can be created and destroyed by the management software as needed. Thus, as far as the guest VMs and hypervisor are concerned the sNICh looks exactly like today's direct-access NICs [34, 15, 24].

The sNICh, however, does not solely operate as a conventional direct-access network interface. Instead, it also integrates the packet processing functions that are typically found in today's data center switches. These functions include packet filtering, packet switching, and packet buffering.

Today's direct-access NICs also support switching packets between virtual machines. But they only provide rudimentary support for advanced switching functionalities like packet filtering. The sNICh extends these devices to support the functionality of a full fledged switch, while enabling a low cost NIC solution by exploiting its tight integration with the server internals. This makes sNICh more valuable than simply a combination of a network interface and a data center switch. Instead, it takes advantage of the server and network proximity to achieve efficiencies that would not be possible if these functions were separated.

The rest of this section explains how packet switching, packet filtering, and packet copying is supported in the sNICh architecture.

3.1 Flow-based Packet Switching

The sNICh uses a flow-based approach to switch incoming packets. A packet is switched in three steps.

- **Flow identification** The packet is parsed to obtain 9 header fields. These header fields include source MAC address, destination MAC address, Ethernet type, VLAN id, source IP address, destination IP address, transport protocol number, source port number and destination port number[1].

[1]The flow is defined here only for a TCP/UDP and IP

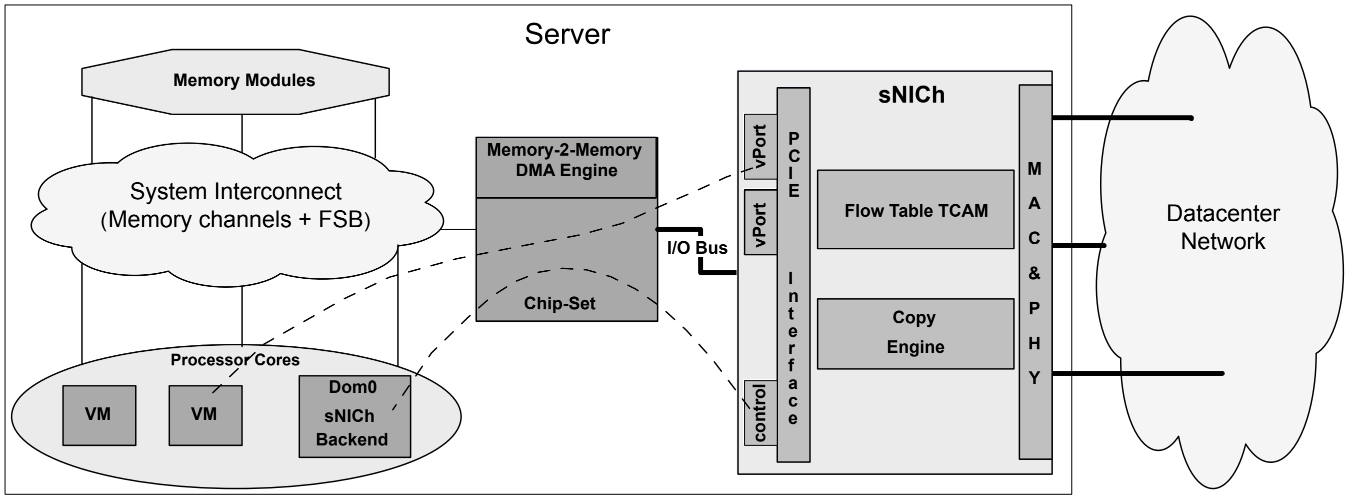

Figure 4: ***sNICh's high-level architecture and its relationship with rest of the server and data center. The sNICh takes the place of a traditional network interface card on the I/O bus. In addition to the standard NIC functionality it also implements a flow-based switch. The sNICh has a flow table, implemented using a TCAM, to cache flow entries. It also has a copy engine that exploits sNICh's direct access to system memory and memory-to-memory DMA engine to achieve high inter-VM throughput, while improving I/O bus availability for traffic to and from the external world. The sNICh backend is implemented in Dom0 software and manages the hardware flow entries.***

These header fields along with the input port form the 10-tuple used to identify a flow.

- **Flow table lookup:** The 10-tuple is used to lookup a flow table to search for matching flow entries cached in sNICh. The flow table is implemented using a TCAM in hardware. Ideally the TCAM size should be large enough to accommodate most of the active flow entries and small enough to be implemented inside sNICh. When the flow table lookup fails (a cache miss), the packet is sent to the sNICh backend software.

- **Flow action execution:** A successful flow table lookup (a cache hit) identifies a flow entry which specifies the action to be performed. Typically, the action is to forward the packet to a particular output port on the sNICh. It can also specify to drop the packet.

A packet which is switched entirely in hardware represents the best-case switching scenario. But packets may have to be handled in software when the flow table lookup fails. The lookup can fail due to two reasons, either the packet belongs to a new flow or the flow entry is no longer cached in hardware. Then the sNICh backend software, which is implemented inside the management domain (Dom0) or the hypervisor, is responsible for correctly forwarding the packet. This is an example of how the sNICh architecture takes advantage of the close proximity of the switching hardware to the server.

When a packet reaches the sNICh backend, it is first filtered using ACL rules. For the sake of clarity, this operation is explained separately in the next section. Then it checks if the packet belongs to a new flow or not. If the packet belongs to a new flow, then it creates a new flow entry along with an action to forward the packet to the appropriate output port. This is possible since the sNICh backend is aware of the MAC addresses and ports allocated to the guest VMs. Then it caches the flow entry in the hardware flow table. If the flow table is full, then the least recently used (LRU) flow is replaced. Finally, the packet is re-injected into sNICh. If a flow entry matching the packet already exists, it simply caches the entry and re-injects the packet. So the sNICh backend maintains all flow entries and caches them in hardware as needed.

The flow-based approach proposed here has similarities to the OpenFlow approach [16]. In particular, our flow identification methodology is inspired by OpenFlow. But the sNICh architecture is not intended to be OpenFlow compatible.

3.2 Packet Filtering

A key function of the modern data center switch is packet filtering. Packet filtering is necessary for functions such as access control list (ACL) processing and quality of service (QoS). These packet filtering operations typically involve multiple hash table lookups

packet. This can easily be extended to accommodate other transport and network protocols.

on simple regular expressions. In modern data center switches this is performed using large ternary content addressable memories (TCAMs). This facilitates high-throughput pattern matching on selected fields in the packet header. But as discussed earlier using large TCAMs to support packet filtering in the sNICh hardware is not a scalable solution. Instead, the packets are filtered in software in the sNICh backend.

In fact even a modern VMM has to filter packets in software in the driver domain or hypervisor, which typically involves list traversal and perhaps some limited hashing. Since most modern systems use TCP segmentation offload (TSO), such packet filtering in software can be done on large packets, instead of MTU-sized packets. This can decrease the filtering burden on the software by an order of magnitude. Even so, performing the necessary lookups in software, on every packet, is prohibitively expensive. It is not possible to approach the filtering throughput of the dedicated TCAM hardware in a data center switch.

But in the sNICh architecture instead of validating every packet, we just validate the flows. Once a flow is validated, subsequent packets belonging to that flow need not be validated and the cost of filtering those packets is not incurred. So even though packet filtering is performed in software, it is not going to be prohibitively expensive.

In the sNICh architecture, the packets are filtered using ACL rules which are installed in the sNICh backend software. Packet filtering is performed in software when the flow table lookup fails and the packet is forwarded to the backend. The packet is matched against all the ACL rules *before* the backend creates a normal flow entry to forward the packet to the appropriate output port and caches it in the sNICh hardware (as explained above).

Access control rules can be of two types. In the first type the default action is to block traffic and the ACL rules specify flows which are allowed. If the packet matches such an ACL rule then it is processed normally. Otherwise, the backend creates a “negative” flow based on this packet, caches it in the hardware, and drops the packet. Now all subsequent packets belonging to this flow will be dropped in the hardware itself.

In the second type the default action is to allow traffic and the ACL rules specify flows which are to be blocked. If the packet does not match any of these rules, then it is processed normally. If the packet matches an ACL rule, then a ”negative” flow entry based on the ACL rule is created and cached in the hardware, and the packet is dropped. Now all subsequent packet flows which match this ACL rule will be dropped in the hardware itself.

Note that “negative” flows are cached in the hardware only when a packet belonging to an active flow is blocked by the ACL rules. Thus we avoid unnecessary utilization of the flow table TCAM entries.

3.3 Packet Copying

After the data center switch has determined the destination port for a packet, the packet must actually be transferred to the output port. In a software switch, this stage requires copying the packet, usually from one address space to another. Such data transfers are typically limited by the memory bandwidth of the machine (as at least one of the source or destination buffers is not resident in the processor caches). This wastes processing resources effectively waiting for memory.

In the sNICh architecture, only the packet headers are copied to the hardware using DMA. These packet headers are then used to perform the switching operations. Once the destination port is identified, the packet is directly copied from the source VM to the destination VM by offloading the packet copy operation.

Modern server platforms enable offloading of data copies using DMA engines. The DMA engine, typically built into the server chipset, can perform asynchronous memory-to-memory movement of data thus freeing up CPU cycles to perform other compute tasks. The sNICh architecture takes advantage of the DMA engine to perform asynchronous packet copies to the destination VM. This is feasible due to the proximity of the sNICh to the server internals. The DMA engine on modern Intel platforms is used as an example here to describe the steps involved in setting up asynchronous DMA operations.

3.3.1 Server DMA Architecture

Recently, Intel introduced Quick Data Technology, as a part of Intel I/O Acceleration Technology (I/OAT), in its server platforms which includes a DMA engine. This technology was originally proposed to eliminate server I/O bottlenecks in native systems by offloading packet copies, from the kernel to application buffers, to DMA engines [11].

The DMA engine presents a standard PCI-E device interface to the host OS. It supports multiple independent DMA channels to the host memory. Each DMA channel has a queue of pending asynchronous transactions associated with it where a transaction describes the operation to be offloaded to the DMA engine. A transaction is setup by creating hardware descriptors in the host memory. Each hardware descriptor includes the source physical address, the destination physical address, and the size of the data to be copied. The transaction descriptors queued for a particular DMA channel are linked together and the physical location of the first descriptor is given to the DMA engine.

Once the new transactions are setup, they are pushed to the DMA engine by tickling one of its registers. This is also sometimes called a “door bell” since it triggers the DMA engine to process the pending transactions.

Then the DMA engine performs the asynchronous data copies without any processor intervention.

3.3.2 sNICh DMA Architecture

The sNICh's proximity to the host will enable it to exploit a DMA engine within the server to accelerate packet copying operations. The sNICh can effectively setup a DMA operation independently of the host to enable packet copying in the background without software intervention or wasting I/O bandwidth. In general, it has been shown that utilizing such DMA engines is only profitable for large copies, as the setup and completion overhead outweighs the benefits [35]. The setup costs arise largely from writes to uncached memory regions to store the DMA descriptors. The completion costs arise largely from polling or interrupt overheads.

The sNICh avoids these overheads in two ways. First, DMA descriptors are stored in memory via *DMA from the sNICh*. Therefore, sNICh can perform these writes in the background and in parallel with other operations instead, whereas the CPU would likely stall waiting for the memory operations to complete. Second, sNICh can easily mitigate the completion overhead by limited polling. After copying a packet using the DMA engine, sNICh will notify the receiving virtual machine that a new packet has been received. As NICs use interrupt moderation to batch notifications to the operating system, sNICh need only poll the DMA engine for completion once every interrupt moderation period, which is frequently $100\mu s$ or more on modern high performance NICs. Therefore, sNICh provides an extremely effective way to exploit server DMA engines to accelerate packet copying among virtual machines.

While not a technical challenge, current server platforms do not allow PCI-E devices to access each other's memory spaces. For sNICh to exploit a server DMA engine, it must be able to directly access the DMA engine in order to initiate the transfer, otherwise the benefits will be lost, as sNICh would have to interrupt the host simply to start the DMA transfer. Furthermore, each sNICh device would need access to its own dedicated DMA channel in order not to incur synchronization overheads with other devices or software that are also attempting to use the DMA engine. Although not supported in current systems, peer to peer PCIe messages needed by sNICh are supported in the PCIe specification and expected to be included in future server platforms.

4. SNICH PROTOTYPE USING SOFTWARE EMULATION

We have built a sNICh prototype where the sNICh hardware is emulated in software. Currently, only inter-VM packet switching by sNICh is emulated. In other words, guest VMs cannot send and receive packets to/from external networks via the emulated sNICh hardware. The sNICh flow table is emulated using hash tables in software (in place of hardware TCAMs). There is an upper limit (1024) on the total number of flow entries in the flow table to emulate TCAM restrictions. Packet copy offloading using DMA engines is also not emulated. The sNICh emulation is run on a dedicated processor core to isolate it from the host software.

We use Xen [2], an open source virtualization platform, as our testbed. The sNICh backend is implemented in Xen Linux Dom0 (management domain) and the sNICh driver in Xen Linux DomU (a paravirtualized guest VM). The sNICh backend is initialized when Dom0 is booted up. This also sets up the control interface which is used for communication between the emulated sNICh hardware and the sNICh backend software in Dom0. The communication occurs out-of-band using a pair of descriptor rings. ACLs are installed in the sNICh backend software using an user-level tool. Packets are matched against the ACL rules in the backend using a simple linear search algorithm.

When a guest VM boots up, it registers with the sNICh backend. During registration, the sNICh backend is informed of the MAC address allotted to that VM. The sNICh backend then creates a new vNIC interface on the sNICh hardware. The vNIC interface implements a pair of descriptor rings for packet transmission and reception. The guest VM (and its vNIC interface) is also bound an unique virtual port (vPort) on the sNICh. Once the registration is complete, the guest VM can directly send and receive packets to/from other guest VMs via the emulated sNICh hardware. Hardware interrupts are emulated using inter-processor interrupts (IPIs). So a guest VM is notified by first sending an IPI to the appropriate processor core and then the hypervisor delivers a virtual interrupt to the guest VM.

Figure 5 explains the steps involved in switching a packet between two guest VMs through sNICh.

5. PERFORMANCE EVALUATION

This section presents a preliminary performance evaluation of the sNICh architecture using the sNICh prototype. In this evaluation, the sNICh architecture is compared with two software switching solutions in Xen, *Linux Ethernet bridge* and *Open vSwitch*. Traditionally, Xen uses the driver domain model [8] to support I/O virtualization. The driver domain is a VM which has direct access to the hardware and performs I/O on behalf of the guest VMs. The driver domain also implements a software switch. The guest VMs send all network packets to the driver domain and the driver domain then switches the packet to send it to another guest VM or to the external network.

The Linux bridge is a software Ethernet switch

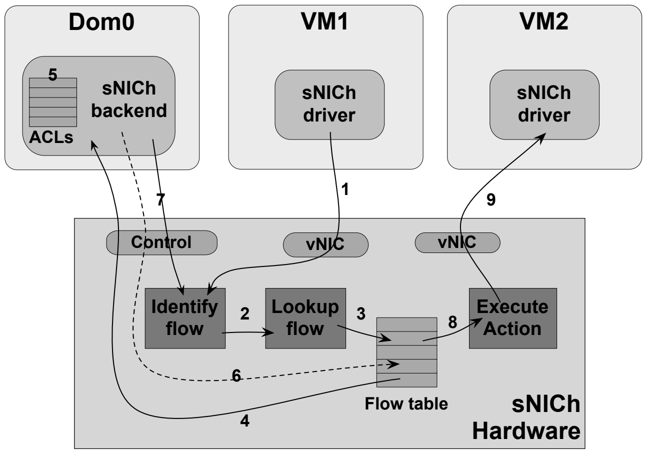

Figure 5: ***Steps involved in switching a packet between two guest VMs through sNICh. (1) sNICh driver in VM1 transmits packet to sNICh hardware through its vNIC interface. (2) sNICh hardware identifies the flow by parsing packet's headers. (3) Flow table is looked up to search for a matching flow entry. (4) If lookup fails, packet is sent to sNICh backend software through control interface. (5) If packet belongs to new flow, backend validates packet against all ACL rules. (6) Flow entry is cached in hardware through control interface. (7) Packet is re-injected into sNICh again through control interface. (8) If flow table lookup is successful, the flow action is executed. (9) Packet is forwarded to VM2 through its vNIC interface.***

shipped with Linux. We use the netfilter/iptables framework (`www.netfilter.org`) in Linux to support ACLs. Open vSwitch [26] is an openflow [16] compatible open source software switch. The openflow approach is similar to the flow-based architecture proposed in this paper where the switching is per-flow instead of per-packet. Open vSwitch implements two network paths: an in-kernel fast-path and an user-level slow-path. The fast-path is implemented as a kernel module which replaces the Linux Ethernet bridge in the driver domain. In the fast-path a software flow table is looked up and the actions associated with the matching flow entry are executed. When the flow table lookup fails, the packets are sent to the control software via the slow-path. Open vSwitch also provides a tool (`ovs-ofctl`) to create ACLs. The ACL rules are initially installed only in the slow-path. When a packet matches an ACL rule in the slow-path, then a flow entry with the appropriate ACL rule actions is added by the control software to the flow table in the fast-path.

5.1 Experimental Methodology

Our experiments are run on an Intel machine with quad-core 2.67 GHz Intel Core i7 processor and 6 GB of memory. The system is configured with up to three guest VMs, each with a single virtual CPU (pinned to a separate processor core) and 1024 MB of memory. The final processor core is used to run either the sNICh emulation or the driver domain. The driver domain configuration is similar to the guest VMs' configuration.

The netperf UDP stream microbenchmark (`www.netperf.org`) is used in the experiments to generate network traffic between the guest VMs. We use five UDP packet sizes (250, 450, 650, 850, and 1050 bytes) in our experiments. The packet throughput at the switch is used as the metric to compare performance. In all the experiments the rate at which the switch processes packets is limited either by the CPU on which the switch is running or the CPU at the transmit side guest VM. We also ensure that there are no packet drops at the switch to avoid distorting the throughput calculations.

5.2 Throughput Results

Figure 6 compares the packet throughput at the three switches when there are no ACL rules to process. We observe that sNICh out-performs both the software switches for all packet sizes. The throughput at both the Linux bridge and Open vSwitch is limited by the driver domain CPU. But in the case of sNICh, the throughput is limited only by the CPU at the transmit side guest VM. Figure 7 shows the CPU cost (in CPU cycles/packet) incurred in driver domain when processing 650 byte packets. Here, the "misc" overheads essentially represent the cost of interfacing with the driver domain to switch packets. This primarily includes the cost incurred in Xen's network backend driver and in Xen's memory sharing mechanism. Clearly, this cost dominates the total packet processing cost in the driver domain.

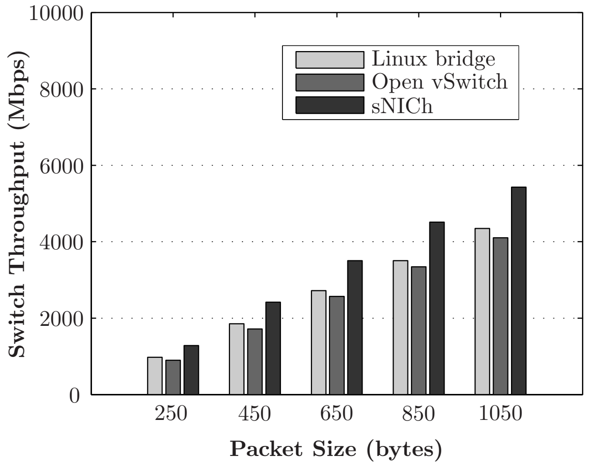

Figure 6: Packet throughput at switch without packet filtering

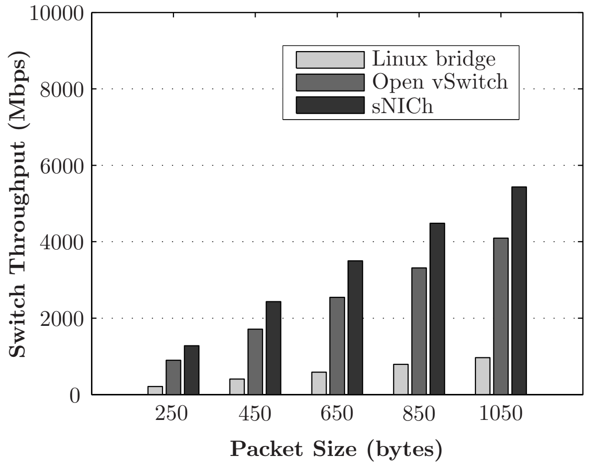

Figure 8: Packet throughput at switch with packet filtering

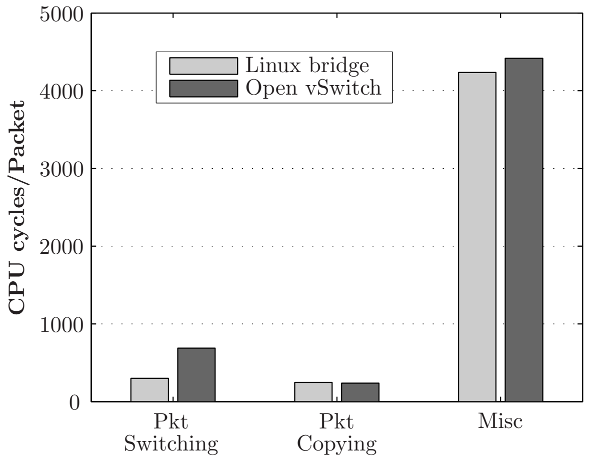

Figure 7: Packet processing cost in driver domain without packet filtering

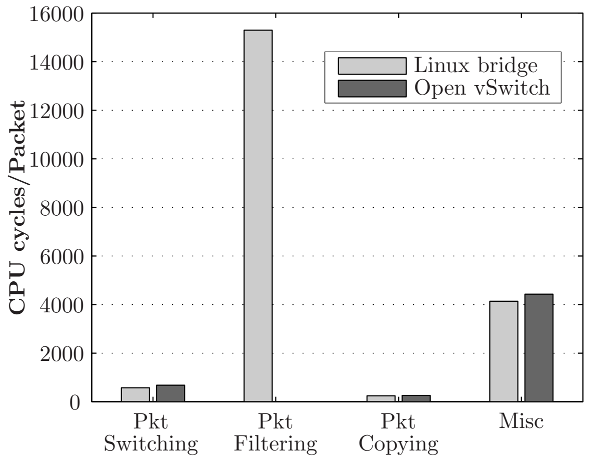

Figure 9: Packet processing cost in driver domain with packet filtering

In these experiments, the performance difference between the software solutions and sNICh is primarily due to the cost incurred in moving the packets from/to the guest VM to/from to the driver domain. This cost is not incurred when a guest VM directly communicates with the sNICh hardware. Note that we are able avoid this cost even in our software emulation due to two reasons. First, Xen's memory sharing mechanism is entirely circumvented. Second, the packet processing within the emulation is simpler and faster than the standard Xen backend driver. Thus we are able to closely emulate the sNICh hardware.

Figure 8 compares the packet throughput when packet filtering is also performed at the switches. Packets are filtering using a single ACL with 971 reject rules in these experiments. The ACL rules are generated using the ClassBench toolkit [31]. Each rule in the ACL filters packets based on five fields in the packet headers (source and destination IP addresses, source and destination ports, and the protocol number) or some subset of these fields. None of the generated packets match any of these rules. We chose this scenario since in most systems, the number of packets dropped by the filter is much smaller than the number of packets that get admitted. We observe that the performance at the Linux bridge drops significantly as compared to the previous set of experiments (Figure 6). This is because the ACL implementation using netfilter/iptables requires every packet to be serially compared against all the ACL rules. The performance of both Open vSwitch and sNICh is not affected since only the first few packets of the flow incur this cost, until the flow is validated against the ACL rules and the flow entry is installed in the flow table. Figure 9 shows the CPU cost in the driver domain when processing 650 byte packets. Clearly, the packet filtering costs dominate the total

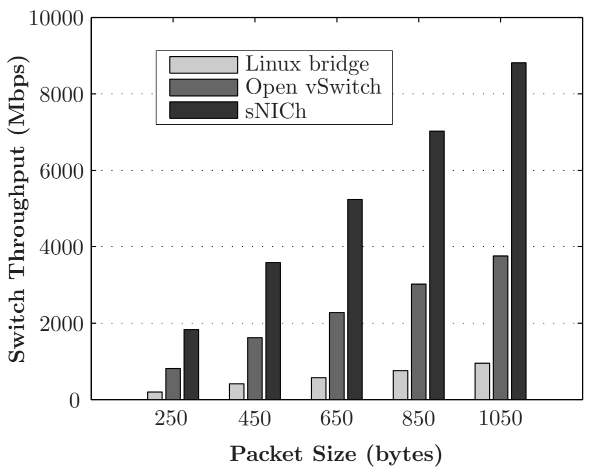

Figure 10: Packet throughput at switch with ACL processing (2 transmit side guest VMs)

packet processing cost in the driver domain with the Linux bridge.

Figure 10 compares the packet throughput at the switches when there are two guest VMs transmitting packets. In the case of sNICh, its throughput nearly doubles for all packets sizes as compared to the previous set of experiments where there is just one guest VM transmitting packets (Figure 8). But the performance of both the Linux bridge and Open vSwitch do not scale. This is because, unlike sNICh, the CPU at the switch is the bottleneck. Clearly, sNICh achieves better scalability than both the software solutions.

5.3 Hardware Vs Software Flow Tables

Open vSwitch implements its flow table in software using hash tables. A fundamental limitation of such a flow table is that it cannot support flow entries with wildcard fields. On the other hand, sNICh can efficiently support flow entries with wildcards as the flow table is implemented using TCAMs in hardware. Wildcards are a very useful feature since a small set of flow entries with wildcards can match a large number of flows. This is essential in a hardware flow table since there is a restriction on the number of flow entries that can be supported.

Typical ACL rules make heavy use of wildcards. For example, an ACL rule which blocks all traffic to a particular port has the IP source and destination fields, among others, wildcarded. While Open vSwitch does support ACL rules with wildcards, it can do so only in the slow-path. When a packet matches an ACL rule in the slow-path (and say gets dropped), then a flow entry is installed in the fast-path to drop all subsequent packets belonging to that flow. A malicious guest VM can exploit this limitation by overwhelming the control software and forcing all packets to take the slow-path. This can be achieved, even if there is an ACL rule blocking this traffic, by sending packets belonging to different flows (different TCP/IP address and ports). Now the slow-path is so overwhelmed with the work caused by the bad packets that the communication channel between the fast-path and the slow-path fills up. Once that happens, legitimate packets that don't have a current matching flow entry get dropped before they even make it to the slow-path to setup a flow entry. We observe this effect when several legitimate TCP connection establishments fail due to the dropping of SYN/ACK packets.

In fact, this limitation is true for any implementation of a flow-based switching approach using a software flow table. Clearly, such a scenario is preventable with sNICh since a small number of TCAM flow entries can effectively block all the illegitimate packets from a malicious guest VM. While this is not proof that sNICh can prevent all malicious traffic, it does show the value of efficient wildcard matching for ACL rules.

6. RELATED WORK

Most of the early work into the network subsystem architectures focuses on accelerating and scaling protocol processing. Makineni, *et al.* have shown that the protocol processing in the traditional network stacks can be too CPU and memory bandwidth intensive to scale to Gigabit speeds [14]. Several designs that are collectively known as offloading approaches address this overhead by moving the protocol processing from the host CPU to the NIC [13, 21].

In spite of the performance benefits, offloading architectures can be too inflexible and hard to evolve and maintain [21]. Hence, an alternative class of proposals, known as *onloading* approaches, retain the protocol processing in the host OS, but supply it with hardware support to make it efficient and scalable. For instance, the coherence attached NIC design proposed in [29] can eliminate interrupt overhead by enabling efficient polling. The *receive side scaling* technique attempts to make the protocol processing scalable to multiple CPUs (or cores) by sending all the interrupts of a flow to the same CPU [20]. Intel's (I/O Acceleration Technology) I/O AT [11] proposes a DMA engine on the system chip set that can be used to eliminate buffer copies in software. The direct cache access scheme [10] reduces latency and saves memory bandwidth, by placing the incoming packets directly into the processor cache (instead of the main memory).

Virtualizing the networking subsystem brings in further overheads [18]. These overheads have been the focus of extensive body of work that can broadly be classified into two categories. The first category retains the virtualization in software and seeks to minimize the overheads [17, 19, 28, 27]. For instance, Ram *et. al.* [27]

propose the use of dedicated receive queues on modern server class commodity NICs to avoid packet copying overheads. On the other hand, the second category of work investigates elimination of these overheads by moving the virtualization support to the NIC itself. For instance, several designs have been proposed to allow a VM to directly access the NIC bypassing the software intermediary [15, 23, 34]. None of these, however, consider any switching functionality, the main focus of our work.

Integration of switch and server functionality was considered in an early InfiniBand NIC, called *InifiniBridge* [6]. This NIC although integrated an InfiniBand switch, it was primarily meant to extend the old PCI (not the PCI-Express of today) and SCSI buses to form a Storage Area Network. RiceNIC [30] and NetFPGA [22] are two extensible NICs with on-board FPGAs that can be utilized to integrate switch functionality. However, such modifications have not yet been attempted on these platforms.

OpenFlow [16] is also a flow-based switching architecture which was primarily proposed as a way to deploy experimental network protocols over existing networks. Existing switches are modified to support the OpenFlow protocol [1] which allows an external controller to program the flow tables within the switch. Thus the controller can be used to setup flow entries which isolate experimental network traffic from real network traffic. The flow-based approach proposed in this paper is inspired by the OpenFlow protocol. In particular, our flow definition is similar to how flows are defined in OpenFlow. Open vSwitch [16] is an OpenFlow compatible software switch for virtualized servers. While the fast-path in Open vSwitch is implemented in software, packets with successful flow table lookups are handled completely in hardware by the sNICh.

Distributing virtual networking across all endpoints within a data center is investigated in [3, 7]. Here software-based components reside on all servers that collaborate with each other and implement network virtualization and access control for VMs, while network switches are completely unaware of the individual VMs on the end-points. Virtual Distributed Ethernet (VDE) is a similar tool that, based on the general concepts of LAN emulation, enables the interconnection of virtual environments via virtual Ethernet switches and virtual plugs [4]. All these approaches differ from our work in that they are designed, at a fundamental level, to be very high-level software implementations and cannot be easily adapted for hardware implementations.

7. CONCLUSIONS

The increasing adoption of virtualization and recent solutions to perform the last hop of the network in software is likely to lead to an untenable situation in the future in terms of overheads. This paper addresses this challenge by introducing the concept and design of a new I/O subsystem called the *sNICh* which is a combination of a network interface and a switching accelerator.

The sNICh utilizes minimal and off-the-shelf building blocks to support key elements of data center switching and leverages its proximity to the server to provide efficient last-hop data center switching. We have built a software prototype of this architecture using software emulation. Preliminary evaluation using this prototype has shown encouraging results. The sNICh outperforms two software switching solutions used in Xen.

We also believe that the increased proximity of the switch and the server can enable other new optimizations and are interested in looking at these further. Overall, as virtualization gets more widely adopted and networking and I/O performance become ever more important, we believe that approaches like ours that examine blurring of the boundaries between the computing and networking infrastructure and their tighter integration, are likely to be a key part of future designs.

8. REFERENCES

[1] Openflow switch specification. version 1.0.0. `http://www.openflowswitch.org/documents/openflow-spec-v1.0.0.pdf`, December 2009.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of SOSP '03*, pages 164–177. ACM, 2003.

[3] S. Cabuk, C. I. Dalton, H. Ramasamy, and M. Schunter. Towards automated provisioning of secure virtualized networks. In *Proc. ACM Conference on Computer and Communications Security*, pages 235–245. ACM, 2007.

[4] R. Davoli. VDE: Virtual distributed ethernet. *TRIDENTCOM '05: Proceedings of the 1st International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*, pages 213–220, 2005.

[5] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Operating Systems Review*, 41(6):205–220, 2007.

[6] C. Eddington. Infinibridge: An Infiniband channel adapter with integrated switch. *IEEE Micro*, 22(2):48–56, 2002.

[7] A. Edwards, A. Fischer, and A. Lain. Diverter: A new approach to networking within virtualized infrastructures. In *Proceedings of the ACM SIGCOMM Workshop: Research on Enterprise Networking*, August 2009. To be published.

[8] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williams. Safe hardware access with the Xen virtual machine monitor. In *OASIS '04: Proceedings of the 1st Workshop on Operating System and Architectural Support for the on demand IT Infrastructure*, October 2004.

[9] A. Greenberg, J. Hamilton, D. A.Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. *SIGCOMM Computer Communcation Review*, 39(1):68–73, 2009.

[10] R. Huggahalli, R. Iyer, and S. Tetrick. Direct cache access for high bandwidth network I/O. In *ISCA '05: Proceedings of the 32nd annual International Symposium on Computer Architecture*, pages 50–59. IEEE Computer Society, 2005.

[11] Intel Corporation. Accelerating high-speed networking with Intel I/O Acceleration Technology. http://download.intel.com/technology/comms/perfnet/download/98856.pdf, April 2007.

[12] J. M. Kaplan, W. Forrest, and N. Kindler. Revolutionizing data center energy efficiency. July 2008.

[13] H.-Y. Kim and S. Rixner. TCP offload through connection handoff. *SIGOPS Operating Systems Review*, 40(4):279–290, 2006.

[14] S. Makineni and R. Iyer. Performance characterization of TCP/IP packet processing in commercial server workloads. In *WWC-6 '03: Proceedings of the IEEE 6th Annual Workshop on Workload Characterization*, pages 33–41, October 2003.

[15] K. Mansley, G. Law, D. Riddoch, G. Barzini, N. Turton, and S. Pope. Getting 10 Gb/s from Xen: Safe and fast device access from unprivileged domains. In *Euro-Par 2007 Workshops: Parallel Processing*, pages 224–233, 2007.

[16] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling innovation in campus networks. *SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[17] A. Menon, A. L. Cox, and W. Zwaenepoel. Optimizing network virtualization in Xen. In *ATEC '06: Proceedings of the USENIX Annual Technical Conference*, pages 2–2. USENIX Association, 2006.

[18] A. Menon, J. R. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoel. Diagnosing performance overheads in the Xen virtual machine environment. In *VEE '05: Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments*, pages 13–23. ACM, June 2005.

[19] A. Menon and W. Zwaenepoel. Optimizing TCP receive performance. In *ATC'08: Poceedings of USENIX Annual Technical Conference*, pages 85–98. USENIX Association, 2008.

[20] Microsoft Corporation. Scalable networking with RSS. http://www.microsoft.com/whdc/device/network/NDIS_RSS.mspx, November 2008.

[21] J. C. Mogul. TCP offload is a dumb idea whose time has come. In *HOTOS'03: Proceedings of the 9th Conference on Hot Topics in Operating Systems*, pages 25–30. USENIX Association, 2003.

[22] J. Naous, G. Gibb, S. Bolouki, and N. McKeown. NetFPGA: reusable router architecture for experimental research. In *PRESTO '08: Proceedings of the ACM workshop on Programmable Routers for Extensible Services of TOmorrow*, pages 1–7. ACM, 2008.

[23] Neterion, Inc. Neterion X3100 series. http://www.neterion.com/products/pdfs/X3100ProductBrief.pdf.

[24] PCI-SIG. Single Root I/O Virtualization. http://www.pcisig.com/specifications/iov/single_root, 2009.

[25] J. Pelissier. VNTag 101. http://www.ieee802.org/1/files/public/docs2009/new-pelissier-vntag-seminar-0508.pdf, 2009.

[26] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker. Extending networking into the virtualization layer. In *HotNets-VIII: Proceedings of the workshop on Hot Topics in Networks*, 2009.

[27] K. K. Ram, J. R. Santos, Y. Turner, A. L. Cox, and S. Rixner. Achieving 10 Gb/s using safe and transparent network interface virtualization. In *VEE '09: Proceedings of the ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 61–70. ACM, 2009.

[28] J. R. Santos, Y. Turner, G. Janakiraman, and I. Pratt. Bridging the gap between software and hardware techniques for I/O virtualization. In *ATC'08: Proceedings of the USENIX Annual Technical Conference*, pages 29–42. USENIX Association, 2008.

[29] M. S. Schlansker, N. Chitlur, E. Oertli, P. M. Stillwell Jr., L. Rankin, D. Bradford, R. J. Carter, J. Mudigonda, N. L. Binkert, and N. P. Jouppi. High-performance ethernet-based communications for future multi-core processors. In *SC '07: Proceedings of the ACM/IEEE Conference on SuperComputing*, pages 1–12. ACM, 2007.

[30] J. Shafer and S. Rixner. RiceNIC: A reconfigurable network interface for experimental research and education. In *ExpCS '07: Proceedings of the workshop on Experimental Computer Science*, page 21. ACM, 2007.

[31] D. E. Taylor and J. S. Turner. ClassBench: a packet classification benchmark. In *INFOCOM '05: Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 2068–2079, March 2005.

[32] VMware, Inc. VMware virtual networking concepts. http://www.vmware.com/files/pdf/virtual_networking_concepts.pdf, 2007.

[33] J. Wiegert, G. Regnier, and J. Jackson. Challenges for scalable networking in a virtualized server. *ICCCN '07: Proceedings of the 16th International Conference on Computer Communications and Networks*, pages 179–184, August 2007.

[34] P. Willmann, J. Shafer, D. Carr, A. Menon, S. Rixner, A. L. Cox, and W. Zwaenepoel. Concurrent Direct Network Access for virtual machine monitors. In *Proceedings of HPCA*, pages 306–317. IEEE Computer Society, 2007.

[35] L. Zhao, L. N. Bhuyan, R. Iyer, and D. N. Srihari Makineni. Hardware support for accelerating data movement in server platform. *IEEE Transactions on Computers*, 56(6), January 2007.