The NIC is the Hypervisor: Bare-Metal Guests in IaaS Clouds

Jeffrey C. Mogul, Jayaram Mudigonda, Jose Renato Santos, Yoshio Turner – HP Labs, Palo Alto

Abstract

Cloud computing does not inherently require the use of virtual machines, and some cloud customers prefer or even require "bare metal" systems, where no hypervisor separates the guest operating system from the CPU. Even for bare-metal nodes, the cloud provider must find a means to isolate the guest system from other cloud resources, and to manage the instantiation and removal of guests. We argue that an enhanced NIC, together with standard features of modern servers, can provide all of the functions for which a hypervisor would normally be required.

1 Introduction

Many people assume that cloud computing ("Infrastructure as a Service," or IaaS) requires the use of virtual machines. In many cases, VMs do offer the best way to manage resources and isolate tenants from each other. The hypervisor layer gives the cloud provider a convenient point to implement its control.

However, Cloud computing does not inherently require VMs. In fact, many customers want clouds to support "bare metal": the guest operating system runs directly on the underlying, unshared server, without any intervening hypervisor [3, 13]. Why do they want bare metal? We know of several reasons (there might be others): because hypervisors reduce *performance*, especially for I/O-intensive applications; because hypervisors also reduce *performance predictability* when they multiplex the CPU, RAM, and network workloads of multiple VMs onto one set of physical resources; because hypervisors create potential *security* holes by expanding the trusted computing base (attack surface) [21]; because *licensing* policies for some widely-used enterprise applications increase costs when virtual machines are used; and because some vendors do not *support* their applications on the hypervisors used by some IaaS providers.

Customers who want to run bare-metal guests in a cloud do give up the cost benefits of multiplexing multiple VMs onto one server, but they see this as an acceptable tradeoff. They still gain several benefits from running their bare-metal guests (BMGs) in a cloud: the cloud provider still does most of the "undifferentiated heavy lifting"[1]; the customer can obtain and release computing resources on relatively short notice, as an "operating expense" instead of as a "capital expense"; and the applications that require a BMG can be co-located in a cloud with other VM-based applications and large datasets, rather than paying the high latency and monetary costs of transferring data into and out of the cloud provider's infrastructure.

Bare-metal guests create a challenge for the cloud provider: even if the provider does not need a hypervisor to isolate the VMs on a single server (since there is only one BMG per server), how does the provider protect the rest of the cloud from these guests? And how does the provider manage a BMG's residency on a server?

We argue that we can replace the necessary hypervisor functions with NIC-based functions, using modest enhancements to existing NIC designs, and exploiting remote-management mechanisms that are standard on modern servers. We refer to NICs that support bare-metal guests as "BMGNICs." While this is not the only approach to supporting BMGs in a cloud, we will argue that BMGNICs have some advantages over the alternatives. In particular, many of the same NIC enhancements are also useful for accelerating the networking functions that are commonly implemented in software switches, for hypervisors that support multiple VMs.

In addition, we see a trend towards integration of NICs into server CPU chips, especially for power-efficient "hyperscale" CPUs that are ideally suited for bare-metal guests. Moving cloud-hypervisor functions into hardware meshes nicely with this trend, because this integra-

[1] This term has been attributed to Jeff Bezos [7]

tion of networking functions is the most critical aspect of eliminating the hypervisor.

1.1 Related work

Szefer *et al.* described NoHype [21], a system which eliminates the hypervisor (after bootstrap) but still supports multiple VMs. NoHype does not seem to include any mechanism that can protect other systems in the cloud network from network misbehavior by a guest. It also requires modifications to the guest OS, which we would rather not depend on. We do not believe NoHype fully solves the problems of supporting BMGs.

2 Refactoring functional responsibilities

In this section, we start by describing the functions that traditional hypervisors and NICs perform, and then we show how one can refactor these functions for a bare-metal guest so that a no-hypervisor platform can perform all of the necessary functions.

Note that we informally use the term "hypervisor" to refer to the entire virtualization software base that runs locally on a cloud compute server node, including the control domain (e.g., Xen's Domain 0) as well as the actual hypervisor layer.

2.1 What does a cloud hypervisor do?

In a cloud with multiple VMs on each server node, the hypervisor has many functions. It:

1. **Instantiates and removes VMs**: When a VM arrives, the hypervisor must create the associated resource bindings, and then boots the guest VM from a system image. When a VM leaves, the hypervisor must clean up.
2. **Allocates CPU and memory resources to VMs**: When multiple VMs share a server's CPU cores and DRAM, the hypervisor must allocate and schedule these resources to specific VMs, and perhaps measure usage for billing purposes.
3. **Isolates VMs from each other, and from itself**: The hypervisor must prevent a VM from directly reading or writing the state of other VMs or of the hypervisor itself, to avoid obvious security and reliability problems.
4. **Provides controlled access to devices**: In non-cloud settings, hypervisors control access to devices such as displays, keyboards, mice, audio, and video. In a cloud, however, the hypervisor only needs to control access to the local disk: space allocation, isolation between the blocks accessed by VMs, and perhaps disk-bandwidth management and disk de-duplication.
5. **Virtualizes the network interface**: When multiple VMs share a physical NIC, the hypervisor must multiplex their outgoing packets and demultiplex their incoming packets. This means rewriting the packets in some way, so that incoming packets can be directed to the right virtual interface (VIF). Popular approaches include VLANs or the VXLAN encapsulation [14].
6. **Virtualizes network interface resources**: The hypervisor might also virtualize NIC resources, such as queue slots and bandwidth, to manage sharing of these resources between VMs.
7. **Virtualizes the network edge switch**: Network virtualization also requires some form of virtual switch, sometimes called a Virtual Ethernet Bridge (VEB), to handle packet delivery. Packets between VMs that reside on the same physical server need not be sent over the physical network; the software switch can deliver them directly to the correct VM. This avoids wasting NIC bandwidth on such packets, and can reduce inter-VM latency. However, this is not always the preferred solution; see §2.3 for details.
8. **Protects other network resources**: Software running on a cloud guest has the potential to create problems for other tenants and for the provider, and must thus be isolated appropriately. In general, this requires both functional isolation, using techniques such as access control lists (ACLs) and network-address virtualization, and performance isolation, using rate limits and perhaps traffic prioritization.
9. **Checkpoints and restores VMs**: Cloud computing offers tenants the ability to flex their resource usage up and down as their workload changes. They may want to suspend a VM during low-load periods, to avoid paying for wasted cycles, and then resume it when the load increases. Checkpoints can also support rapid recovery from failures. The hypervisor handles the process of saving and resuming from checkpoints.
10. **Migrates VMs**: Cloud providers may need to migrate VMs for various reasons, including load balance and hardware maintenance. The hypervisor handles the process of VM migration.

Fig. 1 sketches the networking stack for a typical hypervisor-based cloud. For each VM, the hypervisor supports ACLs, rate limits, and encapsulation/decapsulation (using VLANs, VXLAN, etc.) The hypervisor includes a virtual switch, to demultiplex incoming packets and to switch packets between two VMs on the same server. The figure does not show the mechanisms required to manage other aspects of the server, such as VM instantiation; we assume the use of a framework such as OpenStack.

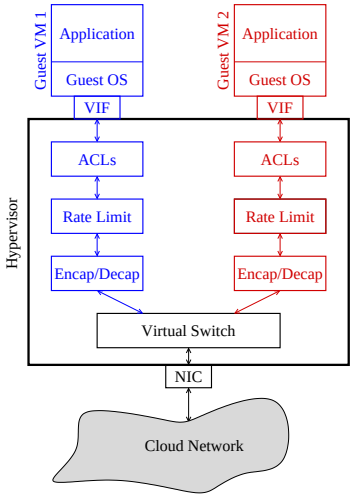

Figure 1: Typical VM-based cloud-network stack

2.2 Cloud-specific NIC functions

The NIC in a cloud computer server not only must send and receive packets, but when the server hosts multiple virtual machines, modern NICs support several functions that improve the efficiency of network I/O.

A hypervisor can, of course, multiplex multiple VMs onto a single NIC by implementing the VEB entirely in software. This works with any NIC, but can add significant CPU overhead.

A NIC that supports "Single Root I/O Virtualization" (SR-IOV) appears to expose multiple virtual PCIe devices for each physical device.

Because an SR-IOV NIC explicitly associates a VM with a set of NIC-level resources, it can offload some forms of QoS support from the hypervisor. Recent proposals, such as *Quantized Congestion Notification* (QCN) [11, 17] require a moderate number of rate limiters in the NIC. (QCN-enabled NICs are already available from NEC, Broadcom, and perhaps others.) And while VXLAN-enabled NICs are just now at the prototype stage [8], we expect cloud-server NICs to support encapsulation/decapsulation soon.

2.3 Virtualization of the edge switch – or not?

At first glance, it would appear that packets between two VMs on the same server could be handled entirely within that server and its hypervisor, by the VEB. However, a surprising amount of work has gone into techniques[2] to switch these intra-server packets *outside* the server, using an external switch. While this seems inefficient, it allows network administrators to manage all network traffic, including intra-server traffic, using a single set of tools. (VEBs are usually invisible to network administrators, and when SR-IOV is used to bypass the hypervisor to gain efficiency, one loses the ability to apply hypervisor-level access-control lists to packet traffic.)

[2]Including VN-Tag (802.1qbh) and VEPA (802.1qbg), and managed virtual switches such as Cisco's Nexus 1000V and IBM's Distributed Virtual Switch 5000V.

2.4 What would a bare-metal platform need?

Use of a bare-metal guest, with no software hypervisor, makes many of the functions performed by hypervisors and modern NICs superfluous. These functions which exist only to support multiple VMs on one server, include: allocating CPU and memory resources, isolating VMs on the same server from each other, and virtualizing the network interface and its local resources.

However, in an IaaS cloud, if there is no hypervisor, some other part(s) of the provider's system would still have to implement many of the functions commonly performed by hypervisors (the numbering here corresponds to that used in §2.1):

1. **Instantiating and removing BMGs**: The provider needs some way to install a new guest on the server, either by creating and booting a local copy of a disk image, or via a network boot. The provider also needs a way to stop the execution of a server, and to boot a "cleanup system" after a guest is evicted.
4. **Providing controlled access to devices**: The provider does not need to perform disk allocation, but it does need a way to erase a disk between different BMGs.
8. **Protecting other network resources**: While a BMG can have full access to local server resources, in an IaaS cloud it cannot be allowed to interfere with other guests (virtual or bare-metal), or with the provider's own infrastructure. Since the network is the point of contact between the BMG and these other entities, the provider still needs a "control point" at which it can encapsulate packets to provided isolated address spaces, enforce ACLs and rate limits, and meter network activity for usage-based billing.
9. **Checkpointing and restoring BMGs**: Just as with VMs, customers have multiple reasons to checkpoint and restore BMGs.
10. **Migrating BMGs**: As with VMs, cloud providers have multiple reasons to migrate BMGs.

Could we move *all* of these functions to an enhanced NIC? Probably not; existing NICs, as PCIe Endpoint devices, are not allowed to restart the CPU. However, by combining a BMGNIC with standard remote-management features (see §3), we can provide bare-metal support.

Function 8 is the most obviously network-related aspect of a BMGNIC, but functions 1, 9, and 10 all require significant (and high-bandwidth) network I/O. The disk-

erase aspect of function 4 requires some means to transmit commands to the disk controller. Functions 1, 4, 9, and 10 require a means for the provider's controller to pause or start the server's CPU at a chosen PC location.

In §3 we will describe a specific system design in which a BMGNIC, in cooperation with standard remote-management support, can support all of the required functions.

2.5 Prior work: sNICh

Before we describe our proposed BMGNIC, we first briefly summarize prior work on *sNICh*, a related system that also pushes functions into the NIC, albeit for support of multi-VM systems.

Ram, Mudigonda, *et al.* described sNICh [18], which refactors how the hypervisor VEB and the NIC divide up the tasks associated with packet-switching. Their main goal was to move most packet-switching tasks from the hypervisor to the NIC, to reduce the performance overheads of a software switch. While a sNICh appears to local software as a NIC, internally it resembles a simple edge switch, incorporating hardware support for OpenFlow-like flow-switching. This allows extension of switch-based network management into the virtual-switch domain, resolving the control vs. efficiency dilemma (§2.3).

sNICh also supports a specialized DMA engine, which allows the NIC to peek at the headers of packets sent by a local VM to determine whether the packet should be copied directly to the buffer of another local VM, or sent out on the NIC's Ethernet port.

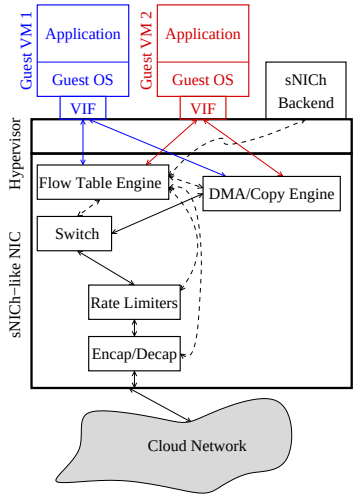

Figure 2: Stack using sNICh-like NIC

Fig. 2 sketches a design that extends the original sNICh approach to include rate-limiting and encapsulation/decapsulation support. Although the original sNICh design was "inspired by OpenFlow [but] not intended to be OpenFlow compatible" [18], we assume a minimally-compatible OpenFlow flow-processing pipeline.

In Fig. 2, the DMA engine allows the flow-table engine to see the headers of packets being sent from local VMs (and also those received from the network, although that datapath is not shown for simplicity). The flow table contains OpenFlow rules, maintained by the sNICh backend process running in the hypervisor (e.g., Xen's Dom0). Since we do not expect a sNICh to have room for many hardware flow-table entries, the backend manages the hardware table as a cache of a much larger software-based table.

When a packet's headers match a table entry, the flow-table engine can instruct (dashed lines, in the figure) the copy engine and switch how to route the rest of the packet. It can also invoke the appropriate rate limiters (as supported in OpenFlow 1.3 via the *meter table*), and by other means in earlier versions of OpenFlow), and encapsulation/decapsulation functions (as supported by OpenFlow's tunnelling mechanism).

Because this approach integrates an OpenFlow-style switch into the server, with the sNICh backend process implementing the switch's side of the OpenFlow control protocol, it allows extension of switch-based network management into the virtual-switch domain. Compared to a purely software approach, such as Open vSwitch, the integration of flow switching with DMA hardware provides higher performance, according to experiments reported in [18].

3 System design

We start by observing that modern server platforms include integrated remote-management engines (RMEs), which allow system administrators or automated controllers to remotely manage many aspects of server operation. These include IBM's *Remote Supervisor Adapter*, Dell's *Integrated Dell Remote Access Controller*, Intel's *Active Management Technology*, and HP's *Integrated Lights-Out* (iLO) technologies. We are most familiar with the details of iLO, so we will focus on HP's mechanism, but we believe that all other major server vendors support the necessary functions, or will soon do so.

RMEs such as iLO were originally designed to support non-virtualized servers, and hence they work even when the server CPU is turned off or is otherwise inaccessible. The iLO design [9, 10] includes a small autonomous CPU and DRAM, power-isolated from the main server, but with the ability to control the server CPU, power and power states, memory, and I/O devices. Recent versions of iLO can communicate with an external controller either via a private NIC, or via the server's own NIC

(via the *Network Controller Side-band Interface* (NC-SI) commonly implemented for server NICs [6]).

For hyperscale systems based on low-power processors (as suggested by [1, 12]), some vendors have embedded RMEs directly on multicore CPU chips (e.g., Applied Micro's SLIMPRO [2]).

3.1 Existing support for BMGs

Many IaaS providers already support bare-metal guests. To the best of our knowledge, they use an integrated management engine to manage the runtime environment on the server, and VLANs to isolate tenants [19].

VLAN support is ubiquitous in modern network switches, which makes this an attractive option for relatively small-scale clouds. However, VLANs do not scale well, since the VLAN tag is limited to 12 bits. Also, it appears difficult to provide customized performance isolation using VLANs. Finally, we expect that most customers will want to combine both BMGs and VMs in their virtual networks, and we therefore would like a solution that supports scalable encapsulation-based isolation (such as VXLAN) mechanisms.

3.2 Our approach

In our design, we refactor the IaaS-provider functions typically performed by a hypervisor, as listed in §2.1, so that:

- a modified sNICh-like NIC protects the rest of the network from the bare-metal guest, including access control, encapsulation, and rate limiting.
- a management engine, such as iLO, handles the installation and removal of BMGs, BMG migration, and BMG checkpoint/restore.
- functions that are superfluous for BMGs, including isolation of server-internal resources, do not have to be implemented on those servers.

We believe our design does not require any changes to a guest operating system, except perhaps for checkpoint/recovery support, which we discuss below.

Fig. 3 sketches the resulting system design, which exploits tight integration between the BMGNIC and RME. One can map the functional blocks of the BMGNIC onto those of the sNICh-like design in Fig. 2 (although *not* onto the original sNICh design, which lacks support for rate-limiting and encapsulation). We move the functions of the sNICh backend into the RME block, since without a hypervisor, there is no other local environment in which to run this processing. The figure shows the RME communicating with the cloud controller via its private NIC and a private management network, but with an RME such as iLO, the RME could instead use the main server NIC via the main cloud network. In either case, the

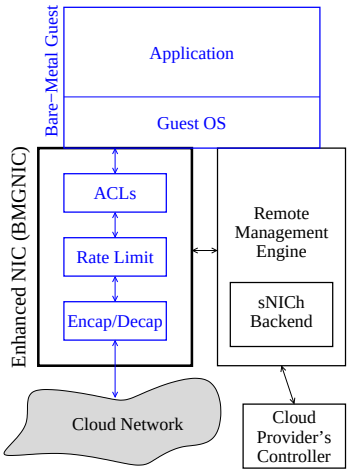

Figure 3: Stack with BMGNIC support

OpenFlow protocol would suffice to manage the network aspects of the server (the sNICh backend corresponding to a switch's control plane, and the BMGNIC to the data plane.) Another protocol (such as in OpenStack) would be used to manage the BMG runtime environment.

The BMGNIC must provide two configuration interfaces: one available to the bare-metal guest, via PCIe, for things that a guest is allowed to change, and another available only to the RME, via NC-SI, for other aspects of configuration. To implement this, we split the BMGNIC's configuration registers into two subsets, one which is accessible to both the guest operating and the RME, and one which is only accessible to the RME.

We believe that checkpoint/restart can be supported with modified iLO software on existing iLO hardware. However, this might require some help from the guest OS, either to support hibernation (as in a laptop OS) or to install a special driver that pre-configures the memory controller to grant iLO DMA access to all of server DRAM. Checkpoint/recovery requires access to storage; iLO supports remote virtual disks, but for higher performance the BMGNIC could include iSCSI support. Migration (but not "live migration") can be implemented by restarting from a checkpoint on a different server.

3.3 Why not do this all in the switches?

One might argue that our NIC-based approach is not necessary, because it is possible to support the network-related aspects of bare-metal guests using only switch-based features, along with traditional NICs. In fact, the argument has been made (e.g., by Davis [5]) that SDN support in switches is sufficient – after all, our BMGNIC design meets our goals precisely because it incorporates OpenFlow mechanisms directly into the NIC.

However, we see several reasons for locating BMG support in NICs rather than in switches:

- The tight integration possible between a BMGNIC and the server's RME can reduce the complexity of the implementation of the provider's entire control system. For example, preserving a guest's MAC address during migration is easy if the RME controls the NIC configuration; it seems much harder to do this entirely at the edge switch.

  Some CPU chip vendors are starting to integrate NIC functions onto CPU chips; for example, Marvell's Armada XP has multiple Ethernet ports [15], and some Intel chips may soon include them [16]. This trend towards on-chip NICs, combined with the integration of RMEs (e.g., SLIMPRO [2]) suggests that tight BMGNIC-RME integration will become easier to exploit in the future.
- Our approach provides a cleaner separation of the network fabric from the network edge, a principle articulated (for different reasons) by Casado *et al.* [4].
- A BMGNIC can be implemented using the same hardware as the enhanced-sNICh design shown in Fig. 2. We believe that sNICh-like NICs provide substantial performance and management benefits when multiple VMs sharing the same server communicate actively, and if the same NIC hardware can efficiently support both sNICh-like functions and BMGNIC functions, this gives the provider considerable flexibility.
- Cloud providers have occasionally experienced long outages due to network misconfigurations. We believe that there are stability advantages to be gained by avoiding frequent reconfiguration operations on switches, even via well-defined interfaces such as OpenFlow. The use of BMGNICs means that no short-term reconfigurations need to be applied to network switches.
- We believe that cloud infrastructures scale better if the resources (TCAM, rate limiters, encapsulation lookup tables, etc.) required to support each additional server come with that server (in this case, as part of the NIC) rather than being pre-provisioned in the switches. This also allows, but does not require, a provider to create a pool of servers that support BMGs without having to upgrade the associated switches.

We admit that none of these reasons are definitive, and that BMGs could probably be supported via SDN switches, with controllers that integrate SDN switches and remote management engines into one seamless environment. However, the BMGNIC approach solves the problems of BMG support with one relatively simple mechanism.

3.4 NIC power and area costs

We would have liked to provide a power and area cost comparison between a traditional server NIC and the BMGNIC design. One could assume that the BMGNIC, for these metrics, would lie between a typical NIC and a low-port-count switch ASIC.

Unfortunately, we have not been able to locate published information on chip areas for such chips. Even power consumption specifications are often covered by non-disclosure agreements. Sohan *et al.* [20] measured active and idle power consumption for a variety of 2010-era 10GbE NIC cards (thereby including PHY power, which has a large effect). They showed that adding an offload engine increases NIC power consumption, but can decrease total system power consumption, so a narrow focus on NIC power might be misleading.

4 Summary

Cloud providers need to isolate their tenants from each other and from the infrastructure. This isolation is largely an aspect of the networking environment: who can talk to whom, and who can consume which resources. While cloud providers have traditionally used hypervisor features to implement network isolation, that approach does not work for bare-metal guests. We have argued that a specialized NIC, or BMGNIC, can provide the necessary isolation, through its integration with the remote management engine located on most modern servers. We have described a specific design for a BMGNIC, which is a modest extension of previously-proposed NIC designs.

References

[1] ANDERSEN, D. G., FRANKLIN, J., KAMINSKY, M., PHANISHAYEE, A., TAN, L., AND VASUDEVAN, V. FAWN: a Fast Array of Wimpy Nodes. In *Proc. SOSP* (2009), pp. 1–14.

[2] APPLIED MICRO. APM "X-Gene" Launch Press Briefing. http://www.apm.com/global/x-gene/docs/X-GeneOverview.pdf, Jan. 2012.

[3] BUTLER, B. New bare metal cloud offerings emerging. *Network World* (May 2012). https://www.networkworld.com/news/2012/050112-bare-metal-cloud-258849.html.

[4] CASADO, M., KOPONEN, T., SHENKER, S., AND TOOTOONCHIAN, A. Fabric: a Retrospective on Evolving SDN. In *Proc. HotSDN* (2012), pp. 85–90.

[5] DAVIS, D. Do we need a network hypervisor for virtualization? http://nicira.com/do-we-need-network-hypervisor-for-virtualization, May 2012.

[6] DISTRIBUTED MANAGEMENT TASK FORCE, INC. (DTMF). Network Controller Sideband Interface (NC-SI) Specification, Jul. 2009.

[7] DRISCOLL, M. Four Lessons for Building A Petabyte Data Platform. http://metamarkets.com/2011/four-lessons-for-a-petabyte-platform/, 2011.

[8] HOFF, B. VMware and Emulex Demonstrate New Network Virtualization Capabilities for Enterprise Data Centers. http://o-www.emulex.com/blogs/labs/2012/08/29/vmware-emulex-demonstrate-new-network-virtualization-capabilities-enterprise-data-centers/, Aug. 2012.

[9] HP. Integrated Lights-Out 3 technology. http://h20000.www2.hp.com/bc/docs/support/SupportManual/c02714903/c02714903.pdf, Feb. 2011.

[10] HP. HP iLO Management Engine technologies. http://h20000.www2.hp.com/bc/docs/support/SupportManual/c03207602/c03207602.pdf, Aug. 2012.

[11] 802.1Qau - Congestion Notification. http://www.ieee802.org/1/pages/802.1au.html.

[12] LIM, K., RANGANATHAN, P., CHANG, J., PATEL, C., MUDGE, T., AND REINHARDT, S. Understanding and designing new server architectures for emerging warehouse-computing environments. In *Proc. ISCA* (2008), pp. 315–326.

[13] LINTHICUM, D. Going native: The move to bare-metal cloud services. *InfoWorld* (May 2012). https://www.infoworld.com/d/cloud-computing/going-native-the-move-bare-metal-cloud-services-192507.

[14] MAHALINGAM, M., DUTT, D., DUDA, K., AGARWAL, P., KREEGER, L., SRIDHAR, T., BURSELL, M., AND WRIGHT, C. VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks, August 2012. https://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-02.

[15] MARVELL. Embedded Processors – ARMADA XP. https://origin-www.marvell.com/embedded-processors/armada-xp/, 2012.

[16] MORGAN, T. P. ARM server hype ramps faster than ARM server chips. *The Register* (2 Jan. 2013). http://www.theregister.co.uk/2013/01/02/arm_server_future/.

[17] PAN, R., PRABHAKAR, B., AND LAXMIKANTHA, A. QCN: Quantized Congestion Notification: An Overview. IEEE 802.1 Interim Meeting, May 2007.

[18] RAM, K. K., MUDIGONDA, J., COX, A. L., RIXNER, S., RANGANATHAN, P., AND SANTOS, J. R. sNICh: Efficient Last Hop Networking in the Data Center. In *Proc. ANCS* (2010), pp. 26:1–26:12.

[19] RUSSKIKH, D. Configuring Bare-metal Switches in OpenStack Cloud Networks: Bare-metal provisioning, part 4. http://www.mirantis.com/blog/configuring-baremetal-openstack-cloud/, Sept. 2012.

[20] SOHAN, R., RICE, A., MOORE, A. W., AND MANSLEY, K. Characterizing 10 Gbps Network Interface Energy Consumption, Oct. 2010.

[21] SZEFER, J., KELLER, E., LEE, R. B., AND REXFORD, J. Eliminating the Hypervisor Attack Surface for a More Secure Cloud. In *Proc. ACM Conf. on Computer and Communications Security* (2011), pp. 401–412.