Accelerated Virtual Switching with Programmable NICs for Scalable Data Center Networking

Yan Luo Eric Murray Timothy L. Ficarra
Dept. of Electrical and Computer Engineering
University of Massachusetts Lowell
Lowell, MA USA
yan_luo@uml.edu
{eric_murray,timothy_ficarra}@student.uml.edu

ABSTRACT

Recently virtual switches in data center hosts have been employed to interconnect virtual machines (VMs) within data center networks. Such a virtual network layer, however, faces performance challenges when the number of VMs and the line rates scale up. Motivated by the performance and programmability of intelligent network interface cards (NICs), we propose to offload the virtual switching onto such programmable NICs (PNICs) to achieve scalable VM networking. We describe the design and advantages of a novel PNIC-oriented data center network architecture. We then present a prototype of a PNIC based virtual switch that supports virtual NICs, OpenFlow switching, clock synchronization and flow monitoring. We finally introduce an efficient packet buffering mechanism enabled by such PNICs and OpenFlow-capable top-of-rack switches for reducing the congestion on network fabric.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; C.2.5 [**Computer-Commu-nication Networks**]: Local and Wide-Area Networks

General Terms

Design;Experimentation;Performance

Keywords

Data Centers, Programmable Network Interface Card, Virtual Switch, Packet Buffering

1. INTRODUCTION

Data centers have become the next-generation computing platforms for enterprises and Internet users. This is primarily due to the economic and technical advantages of resource sharing in data centers. By sharing computing and storage resources through services such as cloud computing or Software-as-a-Service (SaaS), users can amortize the cost of hardware and software. In addition, to ease system upgrades and maintenance, virtual machines are often employed as the ability to migrate virtual machines across the physical hosts results in higher resource utilization. Because of the virtualization of resources, a new virtualized network access layer has been introduced to interconnect VMs within the data centers.

Network virtualization in data centers has recently drawn much attention from the research community as its requirements and opportunities are far different from previously studied areas. The virtualization of the network layer in data center networks (DCNs) is expected to support agility and isolation of VMs. Most physical switches deployed in conventional DCNs are not designed for either supporting such unique VM requirements or flexible enough to augment new functionalities. At the same time, VM networking has inherently unique characteristics, such as the awareness of the migration of VMs and their multicast membership. Therefore, much research has begun exploring the opportunity to introduce a new, flexible and programmable networking layer based on the knowledge of VMs.

This new networking layer is known as a *virtual switch*, and many designs have been proposed including Open vSwitch [2], VMWare's vNetwork Distributed Switch [24] and Cisco Nexus v1000 [11]. These designs implement in-host software-based virtual switches inside either OS kernels or the hypervisors of VMs. In doing so, they leverage CPU cycles available on the hosts to multiplex and control VMs' traffic, instead of relying solely on the dedicated physical switches. While this kind of approach takes advantages of the awareness of VM activities and host events, the in-host virtual switching adds additional workloads to the host CPUs. It also imposes increasing challenges on VM isolation and QoS when per-port line rate reaches 10Gbps and beyond.

We observe another place on VM's packet path where we can employ virtual switching: network interface card (NIC). NICs sit between the hosts and physical switches, and therefore are aware of the VM addressing in the host and have direct access to network fabric. As a result, the NICs are responsive to VM status change and migration. Because of the adjacency, the NICs can also work collaboratively with the top-of-row (TOR) switches in advanced address translation and packet multiplexing. In addition, NICs with processing capabilities are available to accelerate packet processing at high-speed line rates. These devices usually incorporate

VISA 2010, September 3, 2010, New Delhi, India.

programmable processors such as Chelsio's Unified Wire Engine [9], Netronome's NFP-32xx [20], Cavium Octeon [8], Broadcom's BCM57710 [6] and FPGA chips. Such processors make it possible to realize packet switching at the NIC level.

The increasing network processing power and programmability of NICs bring an unprecedented opportunity for disruptive innovations of virtual networking. We are inspired to enrich the networking functionalities of the NICs to improve the efficiency of virtual switches. Our idea is similar to offloading protocol processing from the host CPU to intelligent NICs; we hereby aim to offload the virtual switching from the hosts to the NICs. Our proposed solutions and research directions are significantly distinctive from existing work, most of which concentrate on the either switch-oriented network fabric [3, 12, 18, 13] or host-based virtual switching [2, 24, 11].

In this paper, we first introduce virtual switching and discuss the CPU and memory resource utilization in the hosts. We then propose the programmable NIC based virtual switching architecture which has the benefits of better resource utilization, better isolation of computing and packet switching, and the flexibility to enrich the switching functionalities. Next, we describe the prototype of the PNIC based virtual switch that supports virtual NICs, OpenFlow switching, clock synchronization and flow monitoring. Finally we present a novel packet buffering scheme to alleviate the network link congestion in data centers, leveraging the PNICs and OpenFlow-capable TOR switches.

We make the following contributions in this work:

- We propose a PNIC-centered architecture to carry out virtual switching in the middle ground between the DC hosts and the TOR switches. To the best of knowledge, our design is the first of this kind.
- We present an efficient packet store-and-forwarding mechanism on PNICs for reducing link congestion, taking advantage of the features of emerging OpenFlow switches.
- We make our prototype available as an open source design to enable further research in the virtual networking area.

The rest of the paper is organized as follows. In Section 2 we introduce virtual switch and the programmable NIC oriented DCNs where we shift the virtual network switching from the host to NIC level. We then present our prototype of virtual switch using network processor based NICs in Section 3. In Section 4, we describe in detail a novel packet buffering scheme to alleviate network congestion, relying on the proposed PNIC and OpenFlow-cable switches. We discuss the related work in Section 5. Finally, we conclude the paper in Section 6.

2. VIRTUAL SWITCHING

2.1 Host Based Virtual Switch

Modern data center networks consist of both physical networks connected by switches and virtual networks formed by virtual machines running inside physical hosts. Figure 1 depicts a virtual machine network in a physical host. There can be as many as 120 virtual machines [21] in a DC host,

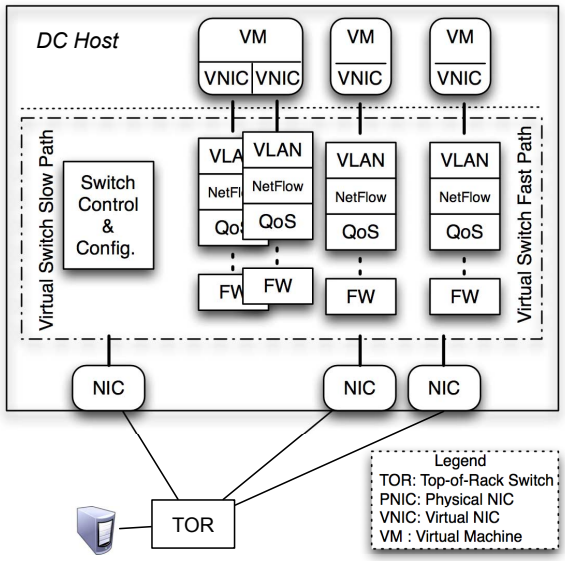

Figure 1: Virtual switch in hosts (figure adapted from Open vSwitch project)

and each VM has at least one virtual NIC (VNIC). The VNICs communicate with external networks through the physical NICs of the host. The traffic multiplexing between the VNICs and physical NICs is achieved with a software layer in the host. This layer of software can be either rudimentary Ethernet bridges (e.g. Linux bridge [1]) or a full-fledged virtual Ethernet switch such as Open vSwitch [2].

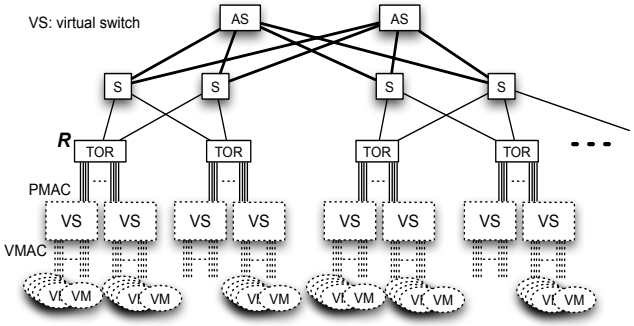

Figure 2: Extended fat-tree network including virtual switches in hosts

A virtual switch (VS) inside a host consists of fast path and slow path components similar to a physical switch. The fast path of the switch includes typical packet processing in a physical switch. Examples include VLAN packet encapsulation, traffic statistics gathering with NetFlow, QoS enforcement and packet forwarding based on forwarding tables. The slow path is designed to support switch configuration and control. For example, a representative software virtual switch, Open vSwitch supports OpenFlow specification [17] so that users can manipulate the forwarding table through OpenFlow APIs.

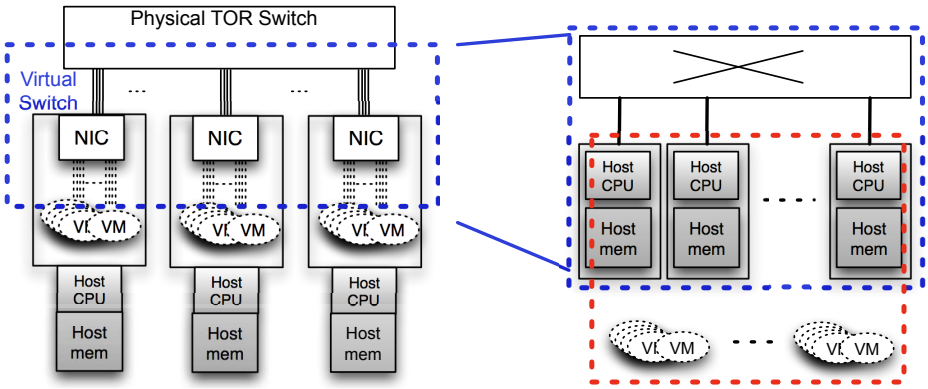

Figure 3: Resource utilization of an in-host virtual switch

2.2 DCN with Virtual Switches

Consequently, the virtual switches at the hosts extend the physical fat-tree network to the hosts, whose topology is illustrated in Figure 2. Each host i in the DCN has np_i physical NICs connected to a TOR switch R. V_i VMs run on host i and nv_i virtual NICs are accessed by the VMs (denoted as $VM(i, y)$ where $0 \leq y < V_i$). The VNICs have their MAC addresses, denoted as $VMAC(i, v)$ where $0 \leq v < nv_i$, while PNICs have MAC addresses $PMAC(i, p)$ where $0 \leq p < np_i$. Similarly, the NICs have associated IP addresses denoted as $VIP(i, v)$ and $PIP(i, p)$, respectively.

2.3 Resource Utilization by Virtual Switches

Figure 3 shows the resource utilization of host based virtual switching. VMs reside in host memory and are executed on the host CPUs. The host CPUs are also utilized by the switching software layer (e.g. Open vSwitch) to multiplex packets between VMs and the VMs outside the host. The host memory is shared by the VMs and the switching software. The regular NICs of the host are connected to the physical TOR switch, providing solely the connectivity to upper layer switching fabrics.

In the design of in-host virtual switches, the management component of the virtual switch is implemented in either kernel or user space, while the fast path of the VS is usually implemented in the OS kernel (e.g. Linux) for performance considerations. The VS maintains the forwarding table and flow statistics inside the kernel, and the packet processing tasks are executed in a kernel thread to reduce the context switching overheads. As the features of the VS grow, the complexity of the VS execution path increases. Note that the VS shares and competes for the host resources (CPU and memory) with the VMs running on the same host. It is unknown how the performance of VS scales as the NIC line rate reaches 10Gbps and beyond. However, it is safe to predict that the hosts can hardly support a scalable number of VMs while dealing with the workloads of packet switching at 10Gbps rate.

Therefore, it makes sense to consider alternatives to the existing host based virtual switching. Figure 4 shows such VS architecture based on programmable NICs (PNICs). We define PNIC as a NIC with programmable processors and memory units such as Netronome NFE-i8000 [19] and NetFPGA [15]. A PNIC often resides in a physical host as a powerful NIC to communicate with external networks. With PNICs, the virtual switch does not have to rely on the host CPU and memory to multiplex packets. Instead, the virtual switch can leverage the resources at the NICs, which can be seen as powerful line cards. This architecture, illustrated in the blue dashed box, resembles the architecture of modern high-end routers, where each line card has the computing power and storage to forward packets independently.

2.4 Benefits of PNIC Based Virtual Switching

We hereby discuss the benefits introduced by the PNIC based virtual switching.

- Better resource utilization at hosts. The host CPUs are dedicated for the VM workloads so that the computationally intensive tasks can obtain a larger share of the CPU cycles. The host memory is also dedicated for VMs rather than being shared by VMs and virtual switches.
- Performance advantages. The fast path processing at the NIC level enables the high speed packet forwarding in a DCN without involving expensive PCIe and DMA transactions. In addition, the NPs are optimized for packet processing thus leading to better packet switching performance than the host CPUs.
- Isolation of computing and packet switching. The natural isolation of VM computing and VS packet processing significantly reduces the context switching overheads and complexity of buffer management. The separate domain (NIC) for packet switching also improves the security and reliability of the DCN.
- Rich functionality. With its programmability, the intelligent NICs can be reprogrammed to incorporate new addressing schemes and routing algorithms.
- Reducing congestion through packet buffering. The memory units available at the programmable NICs form a distributed packet buffer, which can be leveraged to temporarily store packets when certain links are congested. The packets can later be forwarded to destination VMs without retransmission over the higher level inter-switch links, thus balancing the load on the network fabric. We describe the details of such mechanism in Section 4.

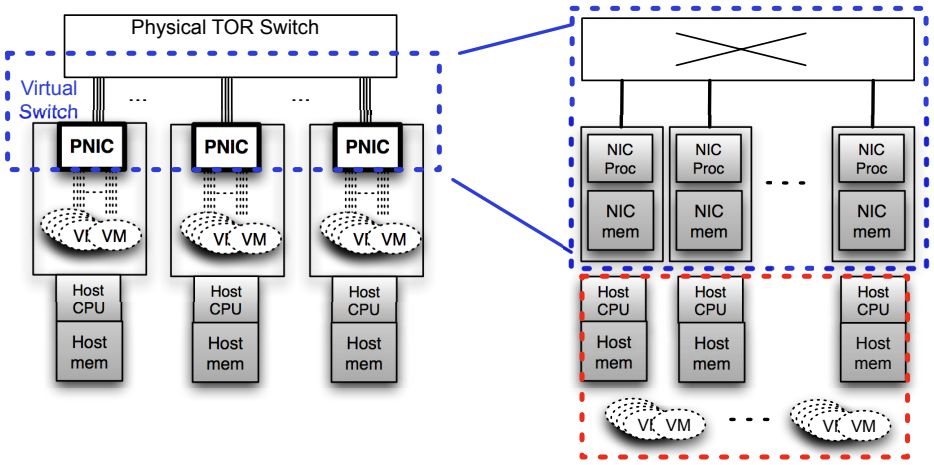

Figure 4: Virtual switch with intelligent NICs

3. VIRTUAL SWITCH WITH PROGRAMMABLE NICS

In this section, we present our preliminary work on implementing virtual switching functionalities onto a programmable NIC.

3.1 Network Processor Achitecture

Programmable packet processors have been incorporated to network interface cards. FPGAs and network processors are the representative processors used for such a purpose. It is expected that such a trend continues as the line rates go up to 10Gbps and beyond. In this section, we first introduce the architecture and characteristics of network processor based intelligent NICs. We then present the design that employs them for accelerating virtual switches.

Network processors (NPs) are designed for high performance packet processing, while offering high programmability with high-level programming languages. An NP, represented by Netronome NFP-32xx [20], usually contains multiple processing cores, co-processors, on-chip memory controllers, and high-speed network I/O interfaces, as shown in Fig. 5. The fast path of packet processing is handled by the programmable cores. Co-processors as well as hardware accelerators are incorporated in many NPs to process control path workload or speed up a particular task such as hashing. An NP is interfaced with both SRAM and DRAM memory modules. Commodity NP-based NICs such as [19] are designed to change the way in which packets are handled in network systems by offloading packet processing from host CPU level to NIC level [16, 25].

3.2 Overview of the PNIC based Virtual Switch

Fig. 6 depicts the overall architecture of the virtual switch based on a programmable NIC. This PNIC is Netronome NFE-i8000 which is equipped with an Intel IXP2855 network processor with 16 cores. Other resources on the PNIC include 40MB QDR2 SRAM, 768MB RDRAM and four 1GbE ports.

The host CPU executes VM workloads and virtual switch software, i.e. Open vSwitch (OVS). The OVS software forwards VM traffic in and out of the host. OVS also communicates with the OVS interface module on the PNIC to configure the NIC level switch. A clock synchronization module on the host periodically sends timestamps to the PNIC for maintaining a high-accuracy clock on the PNIC, which is essential in time-related tasks.

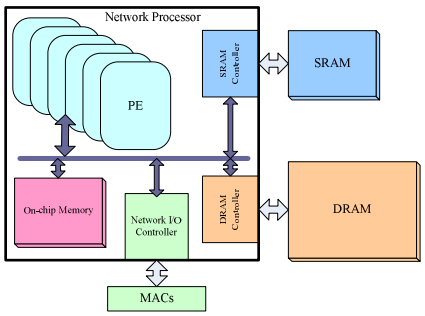

Figure 5: Architecture of a NP.

On the other side of the PCIe bus, we have implemented a set of VS tasks on the programmable cores of the IXP2855 NP, as illustrated in Fig. 6. In particular, the PNIC supports the following functionalities:

- Virtual NICs. Virtual NICs are the logical interfaces to the host for data transfers. We give more details in Section 3.3.
- OpenFlow switching. The fast path of the OpenFlow packet processing is accelerated by the NP on the PNIC. Some details are in Section 3.4.
- Clock synchronization with the host. It is essential that the PNIC maintains highly accurate clock. We rely on the host to provide a reference clock that the PNIC synchronizes to. We describe the design of clock synchronization scheme in Section 3.5.
- Flow monitoring and NetFlow packet generation. We have implemented a flow monitoring function similar to the NetFlow features of commercial switches. Its design is outlined in Section 3.6.

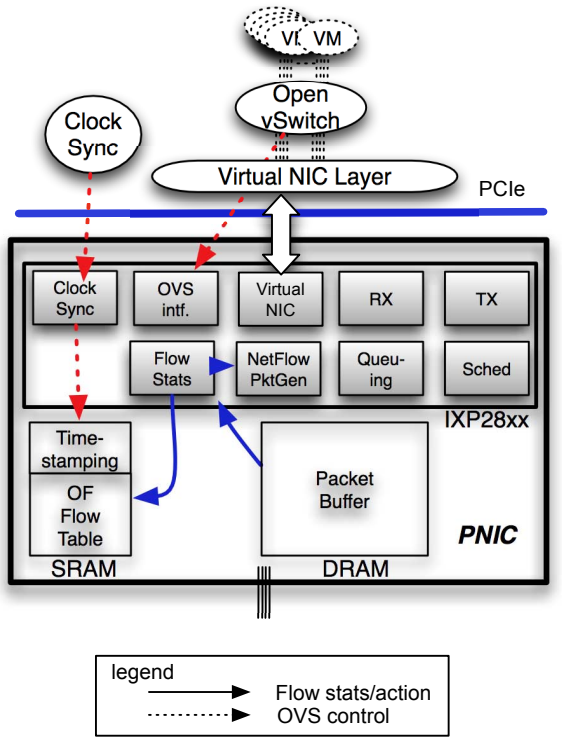

Figure 6: Architecture of virtual switch based on an IXP2855 programmable NIC.

3.3 Virtual NICs

It is important to provide abstrations of the networking resources available on the PNIC. In our design, virtual NICs are presented to the host as regular NICs which have predefined MAC addresses and can be assigned valid IP addresses. A virtual NIC has been developed by using the vendor-provided basic message passing API. While the original design is presented in [7], we extend the work to support more than four virtual NICs. This is done through a VMDq-like design. We maintain a MAC-to-VNIC mapping table (called TBL_{M2V}) in the SRAM on the PNIC. The outgoing packets populate the table with their source MAC addresses and VNIC ID. The destination MAC address of an incoming packet will be used to look up in TBL_{M2V} to determine the destination VNIC of this packet. The packet is tagged with VNIC ID and forwarded to the host CPU, which subsequently delivers the packet to upper layer protocol stack based on the VNIC ID. With TBL_{M2V} and programmable cores on the PNIC, the host CPU is freed from the MAC address lookup, which becomes increasingly complex due to the large number of VMs and VLANs.

3.4 OpenFlow Switching

On the PNIC, we implement the OpenFlow switching algorithm by maintaining a flow table in SRAM. More specifically, we organize the flow entries using a hash table. We calculate the hash value from the 10-tuple OpenFlow fields and use this hash value as an index to access the flow table entry. The actions defined by the flow table entries are applied on the packets accordingly. As shown in our earlier paper [16], the PNIC based OpenFlow switch can reduce the packet round-trip delay by up to 20% while sustaining 4x1Gbps line rate (detailed performance evaluation results are omitted here due to space limit). We also note that there is enough headroom on the programmable cores which can realize other additional processing tasks.

3.5 Clock Synchronization

We leverage the *timestamp* registers on the PNIC and periodic message exchange with the host to synchronize with the reference clock on the host. In such a way, the processing cores on the NIC can perform timing related tasks. The programmable cores on PNIC have timestamp registers which are incremented every 16 cycles (about 11 nanoseconds). The timestamp register is useful to track the time elapse between events. However, the PNIC lacks an onboard real-time clock. To address this issue, we rely on the system clock of the host to maintain clock synchronization, assuming the host runs a NTP-like protocol to keep itself synchronized with an atomic clock. The clock synchronization operation between the PNIC and host is as follows. First, the host sends a special message with its current clock value to the PNIC through a basic message passing API. Then the PNIC recognizes this special "time" packet and extracts the clock time T_h from the packet. Next, a packet processing core of PNIC writes T_h to SRAM and reset the *timestamp* register . Finally, when the time value is needed, the processor reads value t_1 from the timestamp register and calculates the current time T as $T = T_h + t_1$.

3.6 Flow Monitoring

Flow monitoring is an immediate function extension after the PNIC is equipped with VNICs, OpenFlow switching and a synchronized clock. We maintain packet counts in the OpenFlow flow table. The processor core on the PNIC (denoted as "FlowStats" in Figure 6) classifies packets using the 10-tuple headers and look up the flow table using a hash value. The corresponding counters are updated when packets belonging to the flow arrive. Once the counter reaches the predefined threshold, the processor core "FlowStats" notifies another core "NetFlowPktGen", which subsequently generates a NetFlow packet. This NetFlow packet encapsulates the packet counters and a timestamp that is calculated in the way specified in Section 3.5.

We envision the PNIC-based virtual switch can play an important role in data center networks due to its programmability and natural isolation of resources on hosts and NICs. With a close collaboration between the PNIC and the TOR switch, the DCNs can benefit from such an architecture to improve scalability and optimize network resources. In the next section, we present a scenario where the PNIC based virtual switch can enable smart buffering schemes and effectively alleviate the congestion at some hotspots within DCNs.

4. PACKET STORE-N-FORWARD THROUGH OPENFLOW SWITCH AND PROGRAMMABLE NICS

It has been shown that the links near the edges of a data center observe the greatest degree of packet losses due to the bursty traffic among the servers of the data center [5]. While the aggregate traffic rate may be below the link capacity, a momentary traffic burst can result in short-lived congestion.

The traffic ON-period is usually within 100 ms [5], during which the traffic volume reaches up to 100M bits for 1Gbps links, or 1G bits for 10Gbps links. It is feasible to provision a packet buffer (or multiple buffers) to temporarily store the packets when congestion occurs, and later forward them when the congestion goes away.

In this section, we first illustrate a scenario of traffic congestion then propose a solution enabled by PNICs to alleviate the congestion problems. As shown in Fig. 7, let's suppose at time t_1, *all* the physical links of host m are fully loaded. At this moment, the sender virtual machine, $VM(j,0)$, on host j generates a flow of packets destinated to $VM(m,0)$ on host m. The flow has to be stalled at the TOR switch R due to the link congestion at $VM(m,0)$. We try to devise a solution as follows. It may occur that the other virtual switch VS_n, connected to the same TOR switch, has an under-loaded link. Thus, VS_n can sink the flow of packets and buffer them inside host n, as represented with circle (1). VS_n stores the packets in the hope that one of the physical links of host m can be freed in the near future at time t_2. The storage of packets is made possible by the large amount of inexpensive memory available on the hosts. Once a link is freed at at time t_2, the buffered packets will be forwarded from VS_n to VS_m through the last TOR switch, represented with circle (2). The saving of network bandwidth consumption and packet delay is significant because the flow does not congest the higher layer switches or cause packet re-transmission.

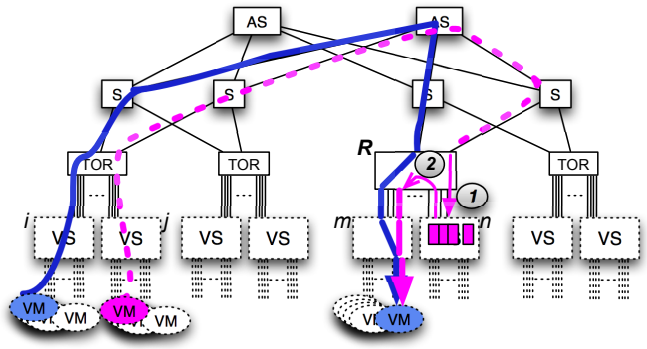

Figure 7: Packet store-and-forward to address congestion problems.

We next describe the hardware architecture for implementing the proposed store-n-forward mechanism. As shown in Fig. 8, PNICs and an OpenFlow-capable TOR switch work collaboratively to realize such a packet buffering scheme. The TOR switch maintains a flow table and is capable of communicating with an OpenFlow controller. The PNIC implements an OpenFlow controller and is equipped with memory for packet buffers.

The proposed scheme has two stages: packet storing stage and packet forwarding stage. The packet buffering protocol for the packet-storing stage is as follows. First, a series of packets arrive at an uplink port of the switch and go to a VM on host i through packet path (a). Such a flow is registered in the flow table at the switch per OpenFlow protocol, and the default action on this flow is to "forward to host i". When the *congestion sensing* unit in the switch detects congestion at the ports to host i, it selects a PNIC that has adequate packet buffer space. This is shown as step 1 in the figure. Such controller selection can follow a round-robin fashion or make assertions based on the availability of memory space on PNICs. Next, the sensing unit notifies the OpenFlow controller at the chosen PNIC (e.g. host k) about the congestion, which is illustrated as step 2 in the figure. The notification message conforms to OpenFlow protocol. Lastly, the OpenFlow controller on the PNIC initiates an OpenFlow command to update the flow entry in the TOR switch with the new action: "forward to host k", as shown as step 3. The packet path now becomes (b) and the subsequent packets are received and stored on the PNIC of host k.

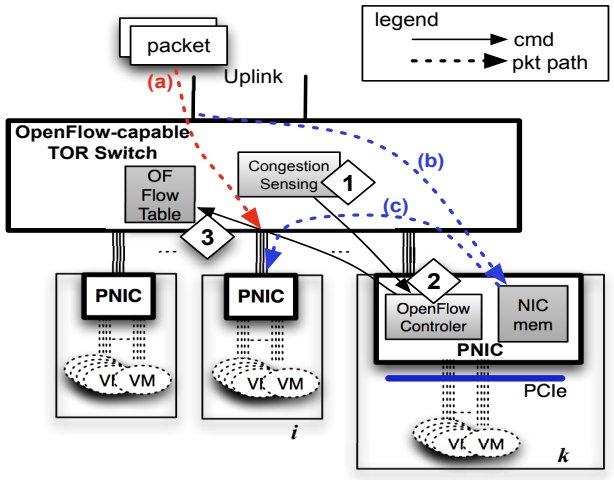

Figure 8: Control procotol of PNIC based packet buffering.

The operations in the packet-forwarding stage are similar to the steps in the packet-storing stage, following basically three steps. In step 1, the congestion sensing unit in the switch detects the congestion status. When the congestion at a port is alleviated, the sensing unit in the switch signals the OpenFlow controller within the PNIC of host k in step 2. The OpenFlow controller sends an OpenFlow command to update again the flow entry in the switch so that the default action on the flow is "forward to host i", as shown in step 3. The processor on the PNIC injects to the TOR switch the previously stored packets, which are subsequently forwarded to their original destination, host i, through packet path (c).

It is worth noting that the PNICs are the better suited than the DC hosts to temporarily store and later forward packets for several reasons: (a) the packets do not have to travel across the PCIe bus to reach the host memory, reducing traffic on the bus; (b) the VMs will be isolated from these packet buffers to ensure security and preserve privacy; and (c) host resources can be dedicated for VMs without the need of reserving resources for packet buffers.

We are in the process of prototyping the proposed packet buffering scheme. There are several research problems that we are currently investigating, such as the selection of the buffers where multiple buffers are present, packet ordering, and load balancing issues. We plan to address these issues in the near future.

5. RELATED WORK

Pfaff et al. presented Open vSwitch (OVS), the first open source virtual switches specifically built for virtual machine environments [21]. Besides fast forwarding performance, OVS supports an external interface for fine-grained control of switch configuration and forwarding behavior. It effectively addresses many challenges of VM networking including isolation in joint-tenant environments, the VM mobility across subnets and so on.

Towards the acceleration of data plane performance for virtual networks, Anwer et al. presented a virtualized data plane based on FPGA [4]. They leverage the open, programmable network processing hardware, NetFPGA [15], to design a hardware-based data plane for virtual networks. Their design is shown to support isolation of virtual routers without compromising the forwarding performance.

In [14], Liao et al. proposed a computing cluster that can perform packet processing in parallel for virtual network substrate. In their design, one or more forwarding machines can be allocated for virtual networks based on the packet processing requirements of the VNs. Their proof-of-concept prototype shows promising results.

Tripathi et al. presented a new virtual NIC (VNIC) based architecture for achieving network virtualization [23]. Their idea is to associate VNICs with dedicated hardware and OS resources. In such a way, the VNIC can ensure full separation of traffic for VMs within the same physical host. This idea has similarity to SR-IOV [22] and VMDQ [10] based designs. While both supporting VNICs, our design differs from [23] in that we leverage the computation power at the NICs to perform complex packet processing.

There exist some commercial products for distributed VM networking. VMWare vSphere vNetwork Distributed Switch provides a centralized point of control for cluster level networking and moves beyond per host network configuration in virtual environments [24]. Cisco Nexus 1000v [11] is a software switch tightly coupled with VMWare's hypervisor. It supports policy-based VM connectivity, mobile VM security and network policies, etc.

Our work is closely related to the Open vSwitch project [21]. The major distinction is that we change the location where virtual switching is carried out. We apply programmable NICs to multiplex VM packets, provide resource abstractions and support management interfaces. To the best of our knowledge, we are the first to propose such virtual switch architecture and design a prototype to evaluate its performance.

6. CONCLUSION

Motivated by the performance and programmability of intelligent network interface cards (NICs), we propose to offload the virtual switching onto such programmable NICs (PNICs) to achieve scalable data center networking. We describe the design and advantages of a novel PNIC-oriented data center network architecture. We then present a prototype of a PNIC based virtual switch that supports virtual NICs, OpenFlow switching, clock synchronization and flow monitoring. We envision an efficient packet buffering mechanism enabled by such PNICs and OpenFlow-capable top-of-rack switches for reducing the congestion on network fabric.

Acknowledgments

This work is supported in part by National Science Foundation award CNS-0709001, a subcontract from the GENI Project Office at BBN Technologies and a grant from Intel Research Council. Eric Murray is supported by the REU supplemental grant from the GENI Project Office. Timothy Ficarra is supported by a National Science Foundation GK-12 Fellowship under award DGE-0841392. The authors thank Dr. Martin Casado from Nicira Networks for his insightful comments on the direction of our work.

7. REFERENCES

[1] Linux net:bridge. http://www.linuxfoundation.org/en/Net:Bridge.

[2] Open vswitch project. http://www.vswitch.org/.

[3] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM*, 2008.

[4] Muhammad B. Anwer and Nick Feamster. Building a Fast, Virtualized Data Plane with Programmable Hardware. In *ACM SIGCOMM VISA'09 Workshop*, Barcelona, Spain, August 2009.

[5] T. Benson, A. Anand, A. Akella, and M. Zhang. Understanding data center traffic characteristics. In *ACM WREN'09 Workshop*, August 2009.

[6] Broadcom. BCM57710 Product Brief: 10-Gbps DUAL-PORT TCP, RDMA, iSCSI CONTROLLER WITH x8 LANE PCI EXPRESS. http://www.broadcom.com/products/Ethernet-Controllers/Enterprise-Server/BCM57710, 2009.

[7] P. Cascon, J. Ortega, W. Haider, A. Diaz, and I. Rojas. A multi-threaded network interface using network processors. In *Proc. of the 17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, February 2009.

[8] Cavium Networks. Octeon multi-core processor family, 2009.

[9] Chelsio. The Unified Wire Engine: Introducing Terminator 3. http://www.chelsio.com/unifiedwire_eng.html, 2009.

[10] Shefali Chinni and Radhakrishna Hiremane. Virtual Machine Device Queues: An Integral Part of Intel Virtualization Technology for Connectivity that Delivers Enhanced Network Performance. Intel White Paper, 2007.

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

[12] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, and P. Pat. VL2: A Scalable and Flexible Data Center Network. In *ACM SIGCOMM*, Spain, August 2009.

[13] Changhoon Kim, Matthew Caesar, and Jennifer Rexford. Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises. In *ACM SIGCOMM*, Seattle, WA, August 2008.

[14] Yong Liao, Dong Yin, and Lixin Gao. PdP: Parallelizing Data Plane in Virtual Network Substrate . In *ACM SIGCOMM VISA'09 Workshop*, Barcelona, Spain, August 2009.

[15] John W. Lockwood, Nick McKeown, Greg Watson, Glen Gibb, Paul Hartke, Jad Naous, Ramanan Raghuraman, and Jianying Luo. Netfpga–an open platform for gigabit-rate network switching and routing. In *MSE '07: Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education*, pages 160–161, San Diego, CA, USA, 2007. IEEE Computer Society.

[16] Yan Luo, Pablo Cascon, Eric Murray, and Julio Ortega. Accelerating openflow switching with network processors. In *ACM ANCS*, 2009.

[17] Nick Mckeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. *OpenFlow: Enabling Innovation in Campus Networks*. The OpenFlow Switch Consortium, mar 2008.

[18] R. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, and V. Subram. PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric. In *SIGCOMM*, August 2009.

[19] Netronome. Product Brief - NFE-i8000 Network Acceleration Card, 2006. http://www.netronome.com/.

[20] Netronome Systems. Nfp-3200 network flow processor product brief, 2009.

[21] Ben Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker. Extending Networking into the Virtualization Layer. In *ACM HotNets*, New York, NY, October 2009.

[22] PCI SIG. Single Root I/O Virtualization and Sharing Specification, Revision 1.0. http://pcisig.com/, 2008.

[23] S. Tripath, N. Droux, T. Srinivasan, and K. Belgaied. Crossbow: From Hardware Virtualied NICs to Virtualized Networks. In *ACM SIGCOMM VISA'09 Workshop*, August 2009.

[24] VMWare. VMWare vSphere: vNetwork Distributed Switch. http://www.vmware.com/products/vnetwork-distributed-switch/, 2010.

[25] L. Zhao, Y. Luo, L. Bhuyan, and R. Iyer. A Network Processor Based Content-Aware Switch. *IEEE Micro*, May-June 2006.