Understanding the PCIe Behavior of Network Cards

Anuj Kalia, Nandita Vijaykumar and Dong Zhou
{*akalia,dongz*}*@cs.cmu.edu, nandita@cmu.edu*

Abstract

Understanding the low-level behavior of network interface cards (NICs) is an important step towards building high-performance networked systems. In this work, we will examine one aspect of this behavior—the interaction between NICs and their PCI Express (PCIe) interface to CPUs. While the PCIe interface to a NIC is often cited as its performance bottleneck [2, 8, 5], the exact steps in the interaction between a NIC and a CPU are not known. A high-level model for this interaction is well known in the networking and systems community [1], but the low level details such as size and number of PCIe transactions, and optimizations such as write-combining are not well understood.

1 Introduction and Related Work

Recent years have seen an industry wide transition from 1 Gbps Ethernet to 10 Gbps Ethernet in the datacenter, and even faster networks with 40+ Gbps of bandwidth are on the horizon. Getting the peak bandwidth from these network devices requires using modified [7] or userspace network stacks [3, 4]. Several recent research projects have used these network stacks for constructing high-performance networked systems [2, 8, 6]. A common problem faced by these systems is that, for small packets (close to minimum-sized Ethernet frames), the performance achieved is smaller than the network's bandwidth. For instance, consider CuckooSwitch [8], that uses dual-port 10 GbE NICs connected to CPU via PCIe 2.0 x8. While the theoretical packet rate per 10 Gbps port with minimum sized packets is 14.88 millon packets per-second (Mpps), CuckooSwitch achieves $\sim$ 23 Mpps per-NIC (the theoretical per-NIC throughput is 14.88*2 = 29.76 Mpps). CuckooSwitch, like other systems [2, 8, 6], explains this discrepancy between expected and observed throughput by claiming PCIe as the bottleneck. However, to our knowledge, there is no work on understanding on *why* PCIe might be a bottleneck. Indeed, a PCIe 2.0 x8 link has 32 Gbps of theoretical bandwidth, and *could* be sufficient to support two 10 Gbps ports.

In this project, we are not challenging the claim that PCIe is the bottleneck for these systems. Our aim is to systematically understand the interaction between NICs and CPUs over PCIe. Consider, for example, the generally accepted model of packet transmission over an Ethernet NIC in terms of the PCIe operations involved.

1. PCIe MMIO write: A CPU-side driver writes to a doorbell register on the NIC.
2. PCIe DMA read: The NIC reads the packet descriptor from the CPU's memory.
3. PCIe DMA read: The NIC reads the packet's data from the CPU's memory.

Apart from checking its accuracy, there are several questions that need to be answered regarding this model. For instance, if the driver wishes to transmit a batch of N packets, how many doorbell writes happen? Are doorbell writes split into multiple PCIe transactions if PCIe "write-combining" is used? Does the NIC read all the descriptors in one DMA read? If so, what is the maximum number of descriptors that the NIC can read in one DMA read?

Our methodology: We propose to answer the above (and other similar) questions by using the PCIe performance counters available in the Xeon series of Intel CPUs. We mainly have access to SandyBridge machines, where two main counters are accessible—The number of *full-cacheline* PCIe reads and writes issued by the device. We believe that more counters, including MMIO write counters and partial-cacheline counters are available on Haswell servers.

2 Deliverables and proposed timeline

The extent of our measurements will depend on the availability of Haswell servers, and on the PCIe counters that are actually accessible on these servers. While we believe that we can get access to these machines soon, some measurements can be done on SandyBridge machines also. For instance, the counters on SandyBridge machines could be sufficient to verify the model of PCIe communication used by InfiniBand cards in HERD [5], and to make some new observations about Ethernet cards. If we do get access to Haswell machines, we will try to answer the questions outlined above including other interesting questions that we can think of.

Timeline: We will dedicate the first one month to understanding PCIe and PCIe device drivers in more detail, and for doing some preliminary measurements on SandyBridge machines. If we don't get access to Haswell machines within this time, we will switch to performing more exhaustive testing on SandyBridge.

References

[1] M. Flajslik and M. Rosenblum. Network interface design for low latency request-response protocols. In *USENIX ATC 13*. USENIX, 2013.

[2] D. Han, A. Anand, F. Dogar, B. Li, H. Lim, M. Machado, A. Mukundan, W. Wu, A. Akella, D. G. Andersen, J. W. Byers, S. Seshan, and P. Steenkiste. XIA: Efficient support for evolvable internetworking. In *Proc. 9th USENIX NSDI*, San Jose, CA, Apr. 2012.

[3] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: a GPU-accelerated software router. In *Proc. ACM SIGCOMM*, New Delhi, India, Aug. 2010.

[4] Intel. Intel Data Plane Development Kit (Intel DPDK). http://www.intel.com/go/dpdk, 2013.

[5] A. Kalia, M. Kaminsky, and D. G. Andersen. Using RDMA efficiently for key-value services. In *Proc. ACM SIGCOMM*, Chicago, IL, Aug. 2014.

[6] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky. MICA: A holistic approach to near-line-rate in-memory key-value caching on general-purpose hardware. In *Proc. 11th USENIX NSDI*, Seattle, WA, Apr. 2014.

[7] L. Rizzo. netmap: a novel framework for fast packet I/O. In *Proceedings of the 2012 USENIX conference on Annual Technical Conference*, June 2012.

[8] D. Zhou, B. Fan, H. Lim, D. G. Andersen, and M. Kaminsky. Scalable, High Performance Ethernet Forwarding with CuckooSwitch. In *Proc. 9th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, Dec. 2013.