Accelerating OpenFlow Switching with Network Processors

Yan Luo[1], Pablo Cascon[2], Eric Murray[1], and Julio Ortega[2]

[1]University of Massachusetts Lowell, USA

[2]University of Granada, Spain

ABSTRACT

OpenFlow switching enables flexible management of enterprise network switches and experiments on regular network traffic. We present in this paper a complementary design to OpenFlow's existing reference designs. We apply network processor based acceleration cards to perform OpenFlow switching. We describe the design options and report our experiment results that show a 20% reduction on packet delay and the comparable packet forwarding throughput compared to conventional designs.

Categories and Subject Descriptors

C.2.6 [**Computer-Communication Networks**]: Internetworking—*Routers*; C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms

Design, Management, Performance

Keywords

OpenFlow, Network Processors, Performance Evaluation

1. INTRODUCTION

Both academia and industry have been investigating the design of clean slate network architecture to address the limitations and ossification of current Internet architecture. GENI, PlanetLab and Emulab are all examples of representative testbeds for researchers to study novel network designs. OpenFlow switch, proposed recently by McKeown et al. in [3], brings programmability and flexibility to enterprise networking by allowing experimentation with heterogeneous Ethernet switches and regular live traffic. The essence of an OpenFlow switch includes flow tables used to implement packet processing (classifying packets and performing actions on them) and the OpenFlow protocol used to manipulate the flow entries. More details of OpenFlow can be found in [2].

Network processors (NPs) are multi-core based processors optimized for packet processing. The unique features of NPs make them candidates for complex network applications. NP-based acceleration cards such as [5] have been designed to change the way in which packets are handled in network systems by offloading packet processing from host CPU level to Network Interface Card (NIC) level. In this paper, we propose a converged architecture to accelerate OpenFlow switching using network processors, describe our design and report experimental results.

We make several contributions in this paper: (1) we present an open source reference design of OpenFlow switching, as an important addition to existing OpenFlow designs. Not only can our NP-based OpenFlow switch be readily deployed in an enterprise network, but also it can serve as an open experiment platform for teaching and experimental purposes. The source code is publicly available at [4]; (2) the work presented here deals with the programming and efficient use of multicore processors (general purpose ones and NPs) and versatile memory components in modern network systems. Nowadays, as technology trends make the multi-core microprocessor as the microarchitecture of choice, this research line is very relevant; (3) we conduct a set of measurement experiments to evaluate the performance of the design and provide interesting insights on the performance. Specifically, we compare the performance of the NP-based OpenFlow switch with OF switches based on dumb NICs and virtual NICs. We also show a 20% improvement of our design on packet delays.

2. THE DESIGN

The OpenFlow implementation on the NP acceleration card consists of two parts. On the host side, the OpenFlow software communicates with the card through a kernel module called *hwtable_nfei8k_mod*. This module supports the passing of OpenFlow commands from the host to the NP, and the values of statistical counters from the NP to the host. This message passing is realized on top of NFD messaging system, a basic PCIe message passing software provided by vendor. The writing of new flow table entries/actions, and the clearing out of existing entries/actions are triggered by the existing OpenFlow switching software (i.e. *secchan.*)

In our design, seven of the 16 microengines (MEs, i.e. programmable cores) on the NP are programmed to perform the following operations: receving, transmiting, queue man-

ANCS'09, October 19-20, 2009, Princeton, New Jersey, USA.

aging, scheduling, packet processing, PCIe read and PCIe write, respectively. Fig. 1 illustrates the organization of the architecture of our design and the pipeline implemented in the NP. We develop software drivers on both the host and the NP to support four virtual NICs, based on the earlier work in [1]. On the packet processing MEs, 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 the 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.

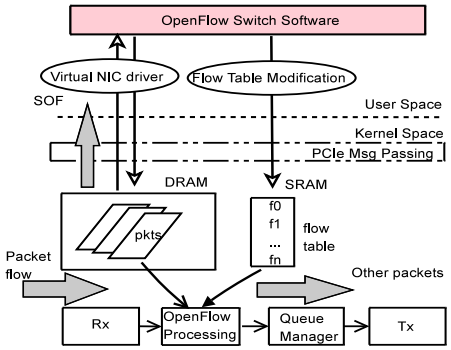

Figure 1: Software architecture of OpenFlow switching on NP

3. PERFORMANCE EVALUATION

We compare the performance of OpenFlow switching on three different architectural options: (1) user-level OpenFlow switch on the host CPU (multi-core x86); (2) kernel-level OpenFlow switch on host CPU (multi-core X86); and (3) kernel-level OpenFlow switch based on NP. The performance metric we focus on is the packet forwarding rate in Mbps.

The host machine we use for the OpenFlow switching is a Supermicro server with dual quad-core processors running at 3.0GHz, 8GB RAM, and three 1Gbps Ethernet NICs. One NIC is used for an out-of-band connection with the OpenFlow controller. The other two are used as two ports of the switch to connect a packet generator and a measurement node, respectively. The packet generator machine runs "packETH", an application that can generate packets with various addresses, protocols and sizes. In particular we generate UDP packets of three different sizes (64, 500, 1500 bytes.) The destination of these packets is the measurement node. We choose six different inter-packet delay times: 20μs, 10μs, 5μs, 2μs, 1μs and the *minimal* delay provided by packETH. We use "tcpstat" as the measurement tool to report the perceived throughput on the receiver side. All the NICs are connected with cross-over cables to eliminate the noise packets (broadcast etc.) normally appearing on a regular switch. Due to space limit, we depict only the results of 1μ inter-packet delay in Fig. 2.

We make several observations on the results. First, the kernel-level switch and NFD-based switch perform consistently better than the user-level switch when dealing small (64B) or medium (500B) packets. This result occurs because of the data copying from kernel to user space in user-level switch software. Such overheads (buffer allocation/deallocation, copying, interrupt) are more significant for small packets

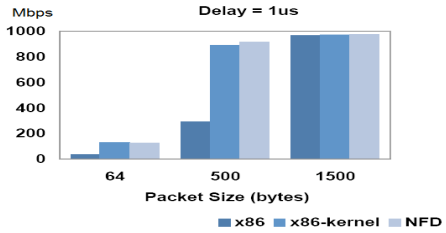

Figure 2: Packet forwarding throughput. (Inter-packet delay = 1μs)

since they occur on a per packet basis. Second, the forwarding rates are about the same for all three designs when the packet is of maximal size. The data-copying overhead from kernel to user space is amortized to large packets. The question is why does the NP not gain on performance since it does not hand over packets to host CPU via PCIe bus. There are several reasons. The NP operates at a much lower frequency (1.4GHz) than the host CPU (3GHz.) Even though the NP saves time on PCIe bus transactions, its forwarding speed is not as high as that of the host CPU. We believe that the newer generation of NPs can perform better. Moreover, the microengines on the NP is not fully utilized. They can be later leveraged to handle concurrent flows (we have only one flow in our current experiment). To test multiple flows will be our future work.

We also measure the round trip time reported by "ping" on the measurement node. This number reflects the delay incurring at the OpenFlow switch of a packet. In this set of experiments we compare the packet delay in three scenarios: OpenFlow with regular NIC, OpenFlow with NP-enabled virtual NICs where the flow tables still reside in the host server, and OpenFlow with NP-accelerated flow table manipulation. The results, illustrated in Fig. 3, show that NP based OpenFlow switch can reduce the packet delay by up to 20% (from 0.313ms to 0.258ms).

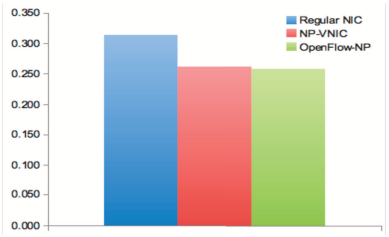

Figure 3: Packet round trip time of "ping".

4. REFERENCES

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

[2] Open Flow Consortium. http://www.openflowswitch.org/.

[3] N. Mckeown et al. OpenFlow: Enabling Innovations in College Networks.

[4] Yan Luo, Pablo Cascon, and Eric Murray. Source code of openflow network processor acceleration module. http://cans.uml.edu/, June 2009.

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