Modularity Meets Batching: Towards an Experimental Platform for High-speed Software Routers

Joongi Kim*†, Seonggu Huh*†, Sangjin Han***†, Keon Jang*†, KyoungSoo Park**, and Sue Moon*
*Dept. of Computer Science, KAIST, {joongi, seonggu, keon}@an.kaist.ac.kr, sbmoon@kaist.edu
**Dept. of Electrical Engineering, KAIST, kyoungsoo@ee.kaist.ac.kr
***Computer Science Division, U.C.Berkeley, sangjin@eecs.berkeley.edu † Students

1 Introduction

Today massively-parallel processors are becoming increasingly popular and cost-effective. Future high performance networking systems design should harness the full computation power of such massively-parallel processors effecitvely for various types of workloads, beyond simple multi-threading on homogeneous fat-core processors. Software routers are no exception. Better yet, parallelism in packet-level processing offers software routers a great opportunity to benefit from parallel SIMT (single instruction multiple threads) processors such as GPUs. We have chosen GPUs among a variety of massively-parallel processors since they are widely available in the market and major vendors provide mature software development kits for free.

Data batching is vital to fully exploit massive parallelism in hardware. Software routers should employ batching at every opportunity and take full advantage of massively-parallel processors. Batching cuts down the per-packet processing cost by removing function call overheads, user/kernel context switching, and expensive I/O interactions. RouteBricks has demonstrated that batching in packet I/O brings much performance improvement in Click [2]. PacketShader shows that batch processing of packets could potentially bring huge performance improvement [3].

Our goal is to build a framework for high-performance software routers. Our framework should make it easy to accommodate technology advances in computing hardware and thus help software routers to scale along. Click is a popular modular software router that implements packet processing in units of modules [5]. Its success has demonstrated the importance of modularity in software router design. Incorporating modularity for programmability and batching for high performance together is a major challenge, for path diversity in per-packet processing interferes with batching. In this work we outline the technical challenges and present our design approaches.

2 Motivating Example

We start with a generic design as in Figure 1. It employs batching at the interface to and from the NICs (network interface card) using the I/O engine from PacketShader in *packs* of packets (a sequential batch of multiple packets). The example configuration has IPv4, IPv6, and IPsec functions and shows packet processing paths among them. Using Figure 1 we elaborate on the technical challenges for our framework:

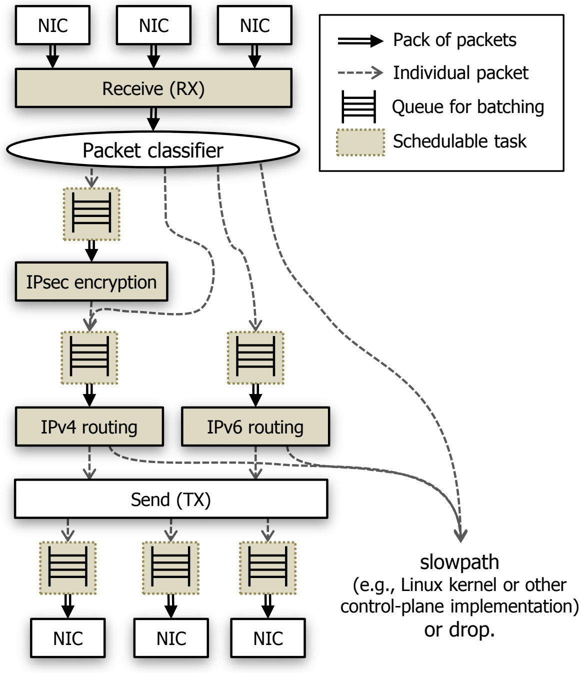

Figure 1: An example of a modular router configuration with batched and non-batched packet flows.

- **Per-packet path diversity within a pack of packets.** Packets arrive at a NIC and the I/O engine aggregates them by the order of arrival. The Receive-Side Scaling (RSS) technique spreads packets evenly over multiple cores. A pack contains one or more flows of packets and they are mixed. Flows may take diverse processing paths and corresponding packets have to be split at output and merged into a pack at input of a module. Here we refer to a *module* as a unit of packet processing without internal path diversity. Splitting and merging between modules incurs performance overhead.
- **Copy overhead between the host and GPU device memory.** Modules can off-load their computations to GPUs. The same portion of a packet could be copied multiple times between disjoint memory spaces, exhausting the very limited PCIe bus bandwidth. We should prevent unnecessary data transfers between GPU and CPU memories.

- **Load balancing and scheduling for overloaded situations.** In general batching increases the average latency. The system should determine whether to process packets immediately or wait for more packets depending on the predicted performance gain. Also a module should decide when to off-load computations to a GPU. If one processor (either CPU or GPU) becomes a hotspot while others are idle, the framework should distribute the work-load.

3 Preliminary Design

We discuss how the framework design copes with the technical challenges respectively, as follows:

- **Efficient pack split-and-merge mechanisms.** Copying from or to the I/O buffer (DMA area) causes about 40% performance degradation for packets larger than 256 bytes in packs with more than 2048 packets/pack. Our implementation in progress addresses this problem by a zero-copy approach using *pointer-of-array* packet queues and packs similar to buffer aggregates of IO-Lite [6].
- **Abstraction of memory resources.** Unnecessary copy prevention requires versioning and version-aware updating mechanisms. We are going to provide two abstractions of data shared with GPUs, *packet buffer* and *table buffer*, differentiated by updating mechanisms and usage. Packet buffers have per-packet states (e.g., version) to keep track of modifications and allow selective copying of only those modified. Table buffers (e.g., used as forwarding tables) provide double-buffering at the GPU side to prevent blocking due to updates.
- **Load-balancing techniques.** (*i*) *Opportunistic off-loading.* Under a light load, CPUs alone are enough to handle all packets. When load increases, packet processing can be off-loaded to GPUs. The framework should decide when to off-load and when not to automatically. (*ii*) *Dynamic module-to-processor assignment.* If a CPU core or a GPU is overloaded due to imbalance in traffic or computation needs by multiple flows, the framework may steal the processing time of other idle processors to disperse workloads.

While we focus on batching for packet forwarding, we should also consider the control-plane integration to manage a router's internal states. We let all control-plane packets bypass the forwarding framework and have the Linux kernel or other user-level applications such as XORP and BIRD handle them [1,4]. They can access the table buffer in order to propagate their internal state changes.

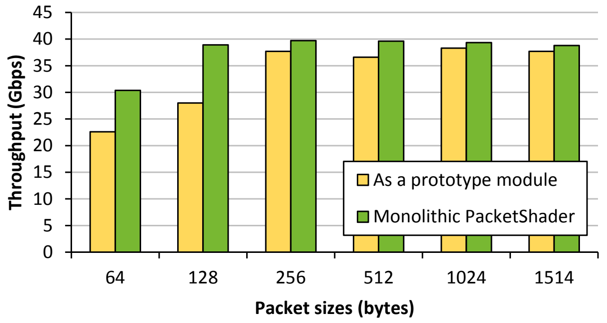

Figure 2: The overhead of preliminary implementation of a packet classifier and input queuing with IPv4 forwarding. About 26% lower performance with 64B packets.

4 Discussion

We have discussed the technical challenges in embracing modularity and batching together and our design approaches. In order to demonstrate the severity of performance degradation we have implemented a simple packet classifier and an input queue for IPv4 route lookup on top of PacketShader. Figure 2 plots the performnace of our preliminary implementation against PacketShader's. Despite that our preliminary implementation of the framework has only added minor changes to PacketShader, we see performance degradation of 26% when the packet size is 64 B.

Figure 2 suggests that modularizing along with batching is a non-trivial job even for a single module and presents more research questions than answered. Will batching keep up with more complex configurations? What is the optimal granularity of modules for both batching and programmability? How far can software routers scale in performance along the modern hardware technology? We plan to continue this work to find the answers by analyzing and refining the design choices.

References

[1] The BIRD Internet Routing Daemon (Web site). `http://bird.network.cz/`.

[2] M. Dobrescu, N. Egi, K. Argyraki, B. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: Exploiting parallelism to scale software routers. In *SOSP*, 2009.

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

[4] M. Handley, O. Hodson, and E. Kohler. XORP: An open platform for network research. *ACM SIGCOMM Computer Communication Review*, 2003.

[5] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Trans. Comput. Syst.*, August 2000.

[6] V. Pai, P. Druschel, and W. Zwaenepoel. IO-Lite: a unified I/O buffering and caching system. *ACM Transactions on Computer Systems (TOCS)*, 2000.