Towards Scalable, Practical Packet Forwarding Plane

Nanako Momiyama
Keio University

Michio Honda
NEC Laboratories Europe

Hideyuki Tokuda
Keio University

ABSTRACT

User-space packet forwarding has demonstrated impressive performance. However, doing so in production has introduced a lot of fundamental and engineering problems, such as isolation, scalability to the number of VMs and API/-CLI compatibility. We therefore explore a software packet switching architecture to address these problems, basing our solution on the kernel packet forwarding plane.

1. MOTIVATION

Software packet forwarding has been an important functionality of a network stack in general purpose OSes. It implements a wide variety of forwarding logics: L2 bridging, IP routing, OpenFlow and middlebox functionality such as filtering, normalizing and proxy. It is used not only to alter hardware switches or appliances, but also, particularly in today's trend of SDN and cloud computing including NFV, to interconnect NICs and virtual machines which could be hundreds of density.

However, a gap between their forwarding performance and ever increasing network capacity, as well as availability of fast packet I/O frameworks like netmap [2] and Intel DPDK, has led community to implementing most of the packet processing logic in user-space. They demonstrated that user-space software switches are faster than the conventional kernel forwarding plane by a factor of ten, which is able to saturate 10 Gigabit Ethernet with minimum-sized packets.

Despite of the superior performance, commercial interests have exposed stresses in using such software switches; examples include lack of compatibility with existing CLIs and APIs, complexity to (securely) serve VMs, scalability to the number of ports or VMs, energy consumption (*e.g.*, due to polling on a NIC) and lack of extensive protocol support.

In this work we therefore do not take a path of user-space networking; we instead explore a kernel forwarding plane architecture that achieves similar performance to user-space networking. We start from improving packet I/O and IP lookup, because they are known to be bottlenecks [2, 3]. However, we show that these improvements only increase packet forwarding rates up to by a factor of 1.8. Since our result suggests other bottlenecks, we run careful measurements to identify them, and sketch our design for further improvement.

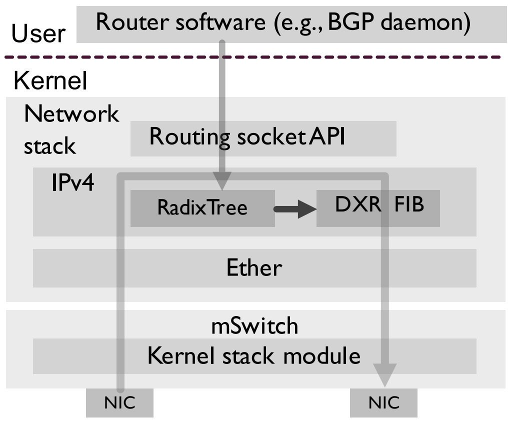

Figure 1: Our router architecture: mSwitch replaces default FreeBSD packet I/O, and DXR builds compact route lookup structures (FIB) based on the radix tree. Overlay arrows indicate control (one from top to bottom) and data (one from bottom to bottom) paths

We primarily focus on L3 IP forwarding because it supports not only the Internet but also data centers as a L2 network does not scale. But our measurements show the problems and proposed solutions apply to other forwarding logics implemented in the OS network stack.

2. IMPROVING PACKET I/O AND IP LOOKUP

Packet I/O: In order to improve packet I/O performance, we must eliminate overheads of dynamically allocating and deallocating packet buffers accompanied with metadata (`mbufs` in FreeBSD, `sk_buffs` in Linux). We also must exploit batching to amortise device access costs. To this end we adopt mSwitch [1], a fast, modular software switch. Since it runs in the kernel, mSwitch is a suitable basis for improving kernel forwarding plane. mSwitch calls a registered callback for every ingress packet, so we register a function that wraps a packet with mbuf which is allocated in the stack area of the code and pushes the packet to the kernel protocol stack. To have mSwitch send packets in a batch, we modify the kernel forwarding plane to intercept the packet before transmitting the packet to the output interface in a usual way. Allocat-

Packet I/O	Lookup	Throughput [Mpps]
Default	Default	1.43
Default	DXR	1.66
mSwitch	Default	1.95
mSwitch	**DXR**	**2.43**

Table 1: IP forwarding performance with different packet I/O and IP lookup methods

ing `mbuf` in the stack area works, because in mSwitch entire packet processing cycle happens within a single function scope. Our approach of using the kernel network stack also enables us to preserve compatibility with existing router software that use socket API (*e.g.*, BSD routing socket), such as a routing daemon.

Route Lookup: Both FreeBSD and Linux use a monolithic routing information database which is managed by a radix tree. This tree is traversed for every packet to find next hop information. A modern architecture for fast IP lookup like DXR [3] create compact routing table structures that includes minimalistic information for packet forwarding, leaving other informations (*e.g.*, aging) in the primary database. We ported DXR which was implemented for old FreeBSD (8.0) to the current version (12.0) of it. We illustrate our system that applies mSwitch and DXR in Figure 1.

3. RESULTS

Table 1 summarises throughput of our FreeBSD router that implements DXR and/or mSwitch. For measurement, we use two machines connected back-to-back. The router machine is equipped with an Intel Core i7-3930K CPU with 3.2 Ghz and an Intel X520 dual-port 10 GbE NIC. Only one CPU core and NIC ring effect in our experiment. The measurement machine is able to send and receive minimum-sized packets at the 10 Gbps line rate.

While the default FreeBSD only achieves 1.43 Mpps, applying DXR for route lookup increases this rate to 1.66 Mpps which is 16% of improvement. Applying both DXR and mSwitch increases the rate to 2.43 Mpps, which is 70% of improvement over the default FreeBSD. While we observe moderate improvements, these forwarding rates are still far away from the 10 Gbps line rate or 14.88 Mpps.

4. IDENTIFYING (NEW) BOTTLENECKS

Having identified that applying known techniques—fast packet I/O and IP lookup—is not enough for fast kernel forwarding plane, we now analyze other bottlenecks. In our router that applies DXR and mSwitch, we insert a `return` statement at various vantage points in the data path; mSwitch is hard-coded to forward packets to an intended output port.

Table 2 summarises results. Our measurements confirm that packet I/O and route lookup are not expensive anymore; mSwitch itself can forward packets at near the line rate, and route lookup with DXR only takes 49 ns. They expose costs

Return function	Description	Throughput [Mpps]	Time Delta [ns]
	No kernel stack	14.36	
`if_input()`	Ethernet input	5.32	118
`ip_input()`	IP input (before route lookup)	4.64	36
`ip_tryforward()`	IP input (after route lookup)	3.66	49
`if_output()`	IP output, ARP resolve	2.43	137

Table 2: Latency breakdown obtained by terminating kernel stack processing at various vantage points and forwarding packets as if it has been done: Ethernet input and output routines are expensive.

at ethernet input and output routines, which take 118 ns and 137 ns, respectively.

5. APPROACH

It is clear that we must avoid `if_input()` and `if_output()` in order to achieve high packet forwarding rates. Complexity in `if_input()` comes from generality of the protocol demultiplexing layer, and that in `if_output()` includes an expensive ARP resolve routine which also involves locks. We would like to bypass these functions, and we are currently extending our router in a following way:

1. mSwitch receives a packet
2. mSwitch performs quick filtering against the packet's destination MAC address and protocol type
3. If the former is a unicast to itself and the latter indicates IPv4, mSwitch directly calls `ip_input()` which is identified as cheap.
4. After route lookup and ARP resolve, the router *caches* the ARP result in the next hop entry in DXR FIB

Since the costs of `if_input()` and `if_output()` are common with other packet processing, such as L2 bridging and firewalling, our solution would also improve performance of such switching logics.

6. CONCLUSION

We believe it is time for bringing recent innovations in user-space networking to the kernel packet forwarding plane in order for scalability, isolation and practicalness. We made significant progress towards this goal by applying fast IP lookup and packet I/O in the kernel. We confirmed moderate effects, but we further analyzed problems remaining. Based on our findings, we sketched new design that decouples generic protocol demultiplex and multiplex layers from fundamental protocol logics, such as IP lookup.

Acknowledgment: We thank Luigi Rizzo for his insightful comments.

7. REFERENCES

[1] M. Honda, F. Huici, G. Lettieri, and L. Rizzo. mswitch: A highly-scalable, modular software switch. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 1:1–1:13, New York, NY, USA, 2015. ACM.

[2] L. Rizzo. netmap: A novel framework for fast packet i/o. In *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*, June 2012.

[3] M. Zec, L. Rizzo, and M. Mikuc. Dxr: Towards a billion routing lookups per second in software. *SIGCOMM Comput. Commun. Rev.*, 42(5):29–36, Sept. 2012.