Accelerating Flow Processing Middleboxes with Programmable NICs

YoungGyoun Moon, Ilwoo Park, Seungeon Lee, and KyoungSoo Park
School of Electrical Engineering, KAIST

ABSTRACT

Software network functions are increasingly popular as they promise operational flexibility unconstrained by physical limitations. However, meeting the stringent requirements of high throughput and low latency in modern networks is often challenging on commodity hardware as network bandwidth is upgraded to 10s of Gbps and the number of concurrent flows grows.

To address the problem, we explore the potential of harnessing modern programmable network interface cards (NICs) to dynamically offload stateful operations of network functions. We identify a set of candidate operations that can be offloaded to programmable NICs, and gauge the expected benefits in terms of CPU load reduction and throughput and latency improvement. We also consider challenges in offloading stateful operations to programmable NICs.

CCS CONCEPTS

• Networks → Middleboxes / network appliances; **• Hardware → Networking hardware**;

KEYWORDS

Flow-processing middleboxes; Programmable NIC; NIC offloading

ACM Reference Format:
YoungGyoun Moon, Ilwoo Park, Seungeon Lee, and KyoungSoo Park. 2018. Accelerating Flow Processing Middleboxes with Programmable NICs. In *APSys '18: 9th Asia-Pacific Workshop on Systems (APSys '18), August 27–28, 2018, Jeju Island, Republic of Korea.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3265723.3265744

1 INTRODUCTION

Modern network functions or middleboxes perform a key functionality in networks. With the trend of "softwarization", network operators can maintain sophisticated network functions freed from the limitations of specific hardware, facilitate a new feature, and dynamically scale middlebox instances in response to changes in incoming workloads.

Unfortunately, achieving high throughput and low latency with software network functions remains challenging on off-the-shelf commodity hardware. Even with user-level networking stacks designed to bypass the inefficient kernel [1, 9, 10], it is often difficult to produce throughputs in excess of multi-10Gbps, and to achieve the latency within hundreds of microseconds [3, 4, 6].

Exploiting programmable NICs to offload CPU computation could be an attractive solution. Equipped with re-configurable network fabrics such as FPGA [5] or specialized flow processors [2, 15], modern programmable NICs allow expressing flexible operations on packet processing at multi-10Gbps speed. Recent works propose to fully offload packet-level or flow-level network functions like in-band network telemetry [12], heavy hitter detection [8], and layer-4 load balancing [14] onto programmable NICs.

In this paper, we consider the potential of harnessing programmable NICs for flow-processing middleboxes, such as layer-7 load balancers or web application firewalls. Stateful operations in flow processing are often deemed as a major source of bottleneck [9], but there have been little effort on accelerating them with NICs. This is largely because such operations have been considered *unoffloadable* as state management is a complex task executed in CPU. In contrast, we identify some opportunities here. First, state management like connection setup and teardown does not require `concurrent` state sharing with host, making it easy for offloading. Second, some operation such as connection splicing is too expensive to run in CPU. NIC offloading of connection splicing would save lots of CPU cycles and memory bandwidth that would be wasted on DMA operations and content copying. Third, certain middlebox operations require only metadata instead of packet content. Programmable NICs could pre-process the packets and deliver only the needed information to host. While there have been prior works [11, 13] that support partial offloading of network functions, our work is the first that identifies the opportunities in flow processing tasks. We explain the opportunities, and share a preliminary result that demonstrates the benefit of NIC offloading.

2 OPPORTUNITIES AND CHALLENGES

We envision potential benefits of NIC offloading in several flow processing applications, and address the challenges in offloading stateful operations.

Connection setup and teardown: We first consider several proxy and gateway applications that actively participate in on-going connections. During connection setup and teardown, existing network stacks handle three or four control packets, and the control packet processing latency often dominates the flow completion time of short-lived connections. Offloading TCP connection management from host to a NIC could save the CPU cycles for DMA operations and control packet processing, and reduce the flow completion time of short-lived connections.

A programmable NIC can handle a TCP connection setup in a stateless manner with a SYN cookie. When a SYN packet arrives at a NIC, it calculates a SYN cookie and responds with a SYN-ACK

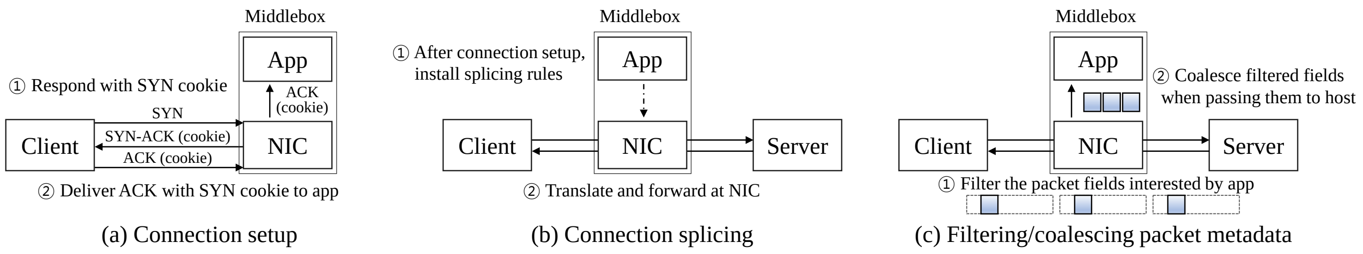
Middlebox
App
① Respond with SYN cookie
ACK (cookie)
SYN
Client
SYN-ACK (cookie)
NIC
ACK (cookie)
② Deliver ACK with SYN cookie to app
(a) Connection setup
Middlebox
① After connection setup, install splicing rules
App
Client
NIC
Server
② Translate and forward at NIC
(b) Connection splicing
Middlebox
App
② Coalesce filtered fields when passing them to host
Client
NIC
Server
① Filter the packet fields interested by app
(c) Filtering/coalescing packet metadata

Figure 1: Example scenarios for NIC offloading of flow processing middlebox operations

packet that reflects the cookie. When the client sends back an ACK packet with a correct sequence number, the NIC informs the host of a new connection, and the host networking stack reconstructs the connection state based on the ACK packet. The host side does not need to handle SYN packets nor need to manage half-open connection states. Offloading connection teardown process to NICs is more challenging because NICs should maintain states, including TCP state and sequence numbers, to track the teardown process. In addition, because a FIN packet can be lost, the NIC should retransmit the packet when it fails to be acknowledged by the other end host.

Connection splicing: Proxy applications including layer-7 load balancer and application-level gateway are used to enforce fine-grained traffic delivery policy based on the request header from a client. When a proxy does not rewrite the content, it can simply fall back to layer-4 forwarding afterwards. For example, many open-source layer-7 load balancers use the splice() system call to transparently forward TCP payload between the two connections in kernel [7, 16]. However, even with splice(), each packet needs to be passed over a PCIe channel twice, incurring heavy DMA overhead. To address it, we dynamically offload connection splicing to NICs. When a proxy decides to splice two connections, it installs forwarding rules on a NIC to translate network addresses and ports between the two connections, as well as to update protocol-specific fields like TCP sequence numbers and acknowledge numbers. After the offload, the NIC dataplane simply performs bidirectional translation while it transparently offloads complex operations such as flow/congestion control and packet retransmission to end hosts.

When the spliced connections finish, the rules for connection splicing must be uninstalled. To handle this, the NIC dataplane should hold minimal states for tracking connection teardown as described above. Layer-7 load balancers often preserve the idle keepalive connections to back-end servers in their cache to reuse them even if the corresponding front-end connection finishes. For connection reuse, we may develop a mechanism that re-stores the connection state from a NIC to the host networking stack.

Filtering/Coalescing Packet Metadata: For middleboxes that inspect a subset of packet header fields (e.g., # of connections, flow size), NICs can extract and pass only those fields to host to reduce the DMA overhead. If multiple packets arrive together, NICs coalesce them into a single packet to minimize the overhead.

3 PRELIMINARY EVALUATION

We demonstrate the benefit of NIC offloading of TCP connection splicing. We build a P4 application that performs connection splicing on a NIC, and run it on a Agilio-CX NIC with a 40GbE port.

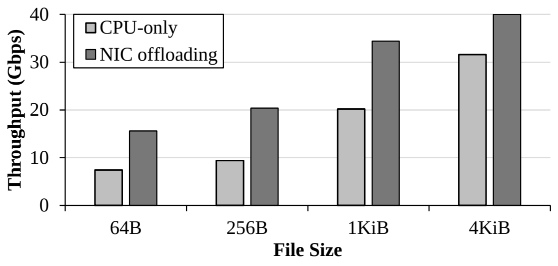

Figure 2: Performance of connection splicing of an L7 proxy (NIC vs. networking stack)

Since the current interface for installing P4 rules on the Agilio NIC is inefficient (< 3000 rules/sec), we evaluate with static rules for TCP connection splicing. We compare its performance against a layer-7 proxy written in mTCP [10], a scalable user-level TCP stack, running on a 8-core CPU (Xeon E5-2640v3) and a commodity 40 GbE NIC (Intel XL710-QDA2). We generate 8,000 concurrent HTTP persistent connections using four pairs of server and client machines each equipped with a 10GbE NIC. Figure 2 shows that NIC offloading of connection splicing outperforms the software-only implementation with an optimized TCP stack, at a much lower cost ($610 vs. $1,390). Our prototype achieves a better response time (835 us vs. 5324 us for 64 B response), avoiding the latency for DMA operations and packet processing.

4 CONCLUSION AND FUTURE WORK

We have explored the opportunities for accelerating flow processing middleboxes with programmable NICs, and our preliminary result is promising. As a future work, we plan to design an optimal offloading policy, instead of blindly offloading the whole flows. We also aim to provide systems support for separating out the offloadable tasks from flow processing middleboxes, and to provide dynamic offloading with a minimal development cost. One possibility is to design and implement a programming interface for seamless offloading to programmable NICs, and to extend the existing networking stacks for flow processing middleboxes [9, 10].

ACKNOWLEDGMENT

We thank anonymous reviewers of APSys'18 for their insightful comments. This work is supported in part by the ICT Research and Development Program of MSIP/IITP, Korea, under projects [High Performance Big Data Analytics Platform Performance Acceleration Technologies Development] and [Development of an ultra low-latency user-level transfer protocol].

REFERENCES

[1] Tom Barbette, Cyril Soldani, and Laurent Mathy. 2015. Fast userspace packet processing. In *Proceedings of the 11th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '15)*.

[2] Cavium. 2018. Cavium LiquidIO II Network Appliance Smart NICs. http://www.cavium.com/LiquidIO-II_Network_Appliance_Adapters.html. (2018).

[3] Byungkwon Choi, Jongwook Chae, Muhammad Jamshed, KyoungSoo Park, and Dongsu Han. 2016. DFC: Accelerating Pattern Matching for Network Applications. In *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16)*.

[4] Daniel E Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. 2016. Maglev: A Fast and Reliable Software Network Load Balancer. In *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16)*.

[5] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. 2018. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18)*.

[6] Rohan Gandhi, Y Charlie Hu, and Ming Zhang. 2016. Yoda: A Highly Available Layer-7 Load Balancer. In *Proceedings of the 11th European Conference on Computer Systems (EuroSys '16)*.

[7] HAProxy. 2018. HAProxy - The Reliable, High-performance TCP/HTTP Load Balancer. http://www.haproxy.org/. (2018).

[8] Rob Harrison, Qizhe Cai, Arpit Gupta, and Jennifer Rexford. 2018. Network-Wide Heavy Hitter Detection with Commodity Switches. In *Proceedings of the Symposium on SDN Research (SOSR '18)*.

[9] Muhammad Asim Jamshed, YoungGyoun Moon, Donghwi Kim, Dongsu Han, and KyoungSoo Park. 2017. mOS: A Reusable Networking Stack for Flow Monitoring Middleboxes. In *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*.

[10] EunYoung Jeong, Shinae Woo, Muhammad Jamshed, Haewon Jeong, Sunghwan Ihm, Dongsu Han, and KyoungSoo Park. 2014. mTCP: a Highly Scalable User-level TCP Stack for Multicore Systems. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI '14)*.

[11] Georgios P Katsikas, Tom Barbette, Dejan Kostic, Rebecca Steinert, and Gerald Q Maguire Jr. 2018. Metron: NFV Service Chains at the True Speed of the Underlying Hardware. In *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18)*.

[12] Changhoon Kim, Anirudh Sivaraman, Naga Katta, Antonin Bas, Advait Dixit, and Lawrence J Wobker. 2015. In-band Network Telemetry via Programmable Dataplanes. In *Proceedings of the Symposium on SDN Research (SOSR '15)*.

[13] Yanfang Le, Hyunseok Chang, Sarit Mukherjee, Limin Wang, Aditya Akella, Michael M Swift, and TV Lakshman. 2017. UNO: Uniflying Host and Smart NIC Offload for Flexible Packet Processing. In *Proceedings of the Symposium on Cloud Computing (SoCC '17)*.

[14] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. 2017. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *Proceedings of the ACM SIGCOMM 2017 Conference (SIGCOMM '17)*.

[15] Netronome. 2018. Netronome: About Agilio SmartNICs. (2018). Retrieved August 3, 2018 from https://www.netronome.com/products/smartnic/overview/

[16] Nginx. 2018. NGINX | High Performance Load Balancer, Web Server & Reverse Proxy. https://www.nginx.com/. (2018).