Advancing Network Function Virtualization Platforms with Programmable NICs

Zhen Ni, Guyue Liu, Dennis Afanasev, Timothy Wood
Department of Computer Science
The George Washington University
Washington D.C., USA
{leonizhen, guyue, dennisafa, timwood}@gwu.edu

Jinho Hwang
IBM T.J. Watson Research Center
New York, USA
jinho@us.ibm.com

***Abstract*—Network Function Virtualization seeks to run high performance middleboxes in a flexible, more configurable software environment. Even with advances such as kernel bypass and zero-copy IO, middlebox platforms still struggle to meet stringent throughput and latency requirements. To achieve line rates as network bandwidths rise, these platforms often must make tradeoffs such as inefficiently dedicating more CPU cores or weakening security and isolation properties. In this paper we explore how advances in programmable "smart NICs" can be leveraged by software middlebox platforms to improve performance, resource efficiency, and security. Our evaluation shows several use cases for smart NICs, which improve performance significantly while reducing resource consumption and providing strong isolation.**

***Index Terms*—Network Functions Virtualization, Isolation, Offloading, Smart NICs**

I. Introduction

Wide area and data center networks are increasingly deploying middleboxes that provide in-network functionality such as security monitoring, cellular connection management, and caching. Prior studies have found that a significant fraction of the devices deployed in data center networks are there to provide different forms of middlebox functionality, not just basic routing and switching. The desire to improve the efficiency and agility of these network services has led to the advent of Network Function Virtualization (NFV), which allows middlebox services to be deployed as software in virtual machines or containers [8], [9].

Achieving high performance for software-based network functions (NFs) remains a major challenge [8]. Current approaches to high performance network functions rely on techniques such as dedicated CPU cores that poll for incoming packets and zero-copy I/O enabled by pre-allocated shared memory. While these techniques allow software platforms to meet line rates of 10Gbps or more, they come at the expense of high resource consumption and reduced isolation between functions [3].

SmartNICs, i.e., network interface cards that offer some form of programmability in the data path, offer a promising new platform to improve the performance of NFV platforms [5]. Similar to graphics processing units (GPUs), SmartNICs contain large numbers of lightweight processing cores which can perform processing on packets in parallel before the packets are DMA'd from the NIC into the host's main memory to be processed by the CPU. The programming models for SmartNICs are still being developed due to the hardware's nascent state, but several options are currently available, ranging from network-specific languages like P4 to fully customizable C functions [1].

In this paper we explore how programmable NICs can be used to optimize an NFV platform. We demonstrate how they can be used to offload some aspects of an NFV framework to higher performance hardware, while still allowing flexible, software-based control. Unlike traditional static NICs, the SmartNICs are programmable. We are able to keep the advantage of NFV while executing network workloads on the SmartNICs. In particular, we evaluate how a SmartNIC can:

- Improve NF security and platform efficiency by routing to isolated memory pools from the NIC hardware instead of requiring dedicated CPU cores.
- Offload computation to the NIC to reduce CPU load and improve latency, in some cases bypassing the host's software stack entirely.

We evaluate these approaches using a Netronome Agilio CX network card, using both its P4 [4] and C-based programmability [1]. We show how this can be integrated with the OpenNetVM NFV platform to improve throughput by 2 million packets per second and reduce latency by at most 8 μs compared to a pure-software technique.

II. Challenges and Opportunities

As network speeds rise and network/cloud operators devise more functionality to embed in the network, high performance network function virtualization becomes increasingly important. However the state-of-art for NFV is still inadequate. There is no perfect NFV infrastructure that could provide high performance, rapid development, and strong security guarantee simultaneously. Many NFV platforms use "zero-copy" packet processing to achieve line-rate throughputs and low latency with commercial off-the-shelf (COTS) servers [3], [8], [13], [17]. However, requiring dedicated cores to poll packets from the NIC has limited their scalability and using a shared global memory pool for DMA'd packets has left potential threats for network eavesdropping from neighboring NFs. In this section,

978-1-7281-1434-7/19/$31.00 ©2019 IEEE

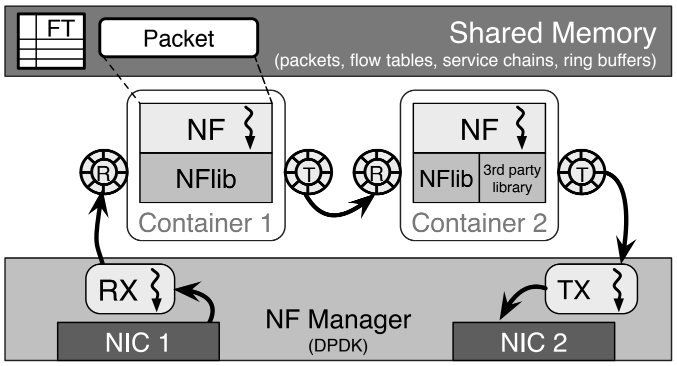

Fig. 1. OpenNetVM Platform Architecture

we discuss how the recent emergence of SmartNIC can help address these challenges.

A. High Performance, Secure, Efficient NFV

Ideally, the network data plane should provide the flexibility and convenience of a cloud infrastructure: operators should be able to easily deploy functions, scale them on demand, and ensure their integrity and isolation.

Security: Many current NFV platforms pursue high performance but trade off security concerns [11], [17]. High performance in network function service chains is commonly achieved with zero-copy I/O. An example NFV architecture is shown in Figure 1. OpenNetVM [17] is a high-performance NFV platform for researchers and developers to easily deploy network services. Network functions can be deployed as Docker Containers to run the code along with its dependencies. The NF Manager handles receiving and sending packets from the NIC ports, and it uses the Data Plane Development Kit (DPDK) to efficiently bypass the kernel stack and load packets into a global shared memory region in user space. This memory region is also shared among all NFs so that NF communication can be optimized to just pass along packet descriptors instead of copying packet data. Although this approach can achieve high throughput and low latency, it also brings security risks since all packet data is in one shared memory pool that can be manipulated by malicious neighbors.

Efficiency: Performance is also achieved at the expense of resource efficiency. DPDK-based applications typically rely on threads that use polling to check for the arrival of messages, i.e., new packets. Polling eliminates the high cost of interrupt processing, but it keeps CPU cores fully loaded at 100% regardless of the incoming arrival rate. As the number of NFs rises or the management layer complexity grows, the required number of CPU cores continues to grow. For example, in OpenNetVM, the management layer can act as a mediator between communicating NFs. This allows the management plane to redirect packets and ensure only NFs with the appropriate communication permissions can communicate. While this is a useful feature, the necessity of polling by management cores means it comes at a high resource cost.

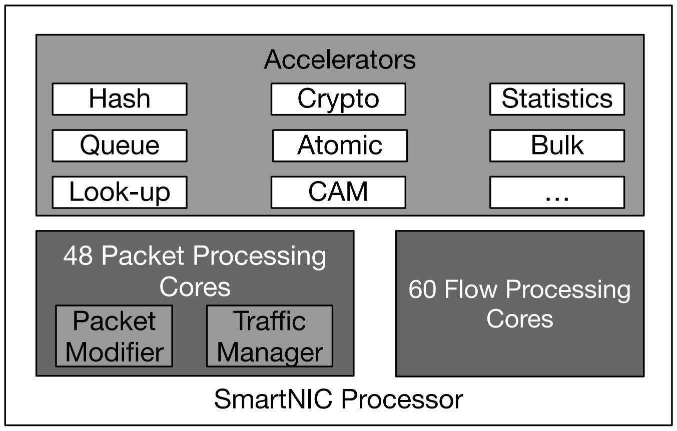

Fig. 2. Processor of SmartNIC

B. Opportunities

In recent years, cloud datacenters need to serve hundreds of thousands of users and that number is still increasing fast. Because of the growth of tenants, VMs, and applications, programmable, virtual networking has been developed into reality. Service providers allow users to bring their own network and security policies, as well as program in custom functionality. However, the increasing demand for CPU cores and power is becoming more and more unacceptable as network line rates rise. Programmable SmartNICs are purpose-built to solve the performance and scaling challenges.

SmartNIC HW: In this work we consider the Agilio SmartNICs, developed by Netronome [1]. A SmartNIC, also called a programmable NIC, allows engineers to build networking accelerator or dataplane applications into the NIC hardware [15]. Figure 2 shows the high-level structure of Agilio NFP-4000 SmartNIC processor. The processor includes 48 packet processing cores and 60 flow processing cores (up to 120 cores in NFP-6000 SmartNIC processor), which guarantees high performance while offloading processing-intensive functions such as virtual switch, load balancing, and security. The packet processing cores are used for the basic functionality such as rule matching, packet modifications, and so on, and the flow processing cores are programmable blocks that can run custom P4 and microC programs. While traditional network cards have only basic control plane functionalities, the SmartNICs also have over 60 hardware accelerators for deep packet inspection (DPI) which support hash, cryptography, statistics, and more. These accelerators combine the flexibility of software and the performance of hardware. The programmable processing cores are fully assisted by the hardware accelerators for faster computation.

SmartNIC Programming: FPGA-based NICs have been available for some time, but they require a difficult programming model not familiar to most network operators. Newer SmartNICs have expanded high level and low level programming models [1], [2]. The network flow processor (NFP) can be programmed for the custom packet/flow processing using P4 and C languages. P4 is a high-level language for designing packet processing services [7]. Unlike OpenFlow, which requires protocol headers to be specified in the language itself, P4 is protocol independent, allowing a wider range of functionality to be implemented. P4 programs define a programmable match-action pipeline that allows customizable programming for each incoming flow [10]. C-based SmartNIC programs provide even greater flexibility in the functionality that can be implemented, but require a more detailed understanding of the underlying HW API.

NFP-based Agilio SmartNICs support the following programming models:

- Host API-based Programming Model
- User Datapath Programming Model

The high level host API-based programming model supports network I/O configuration, and standard datapath features, but lacks complete customizability. This programming model is suitable for users who want to utilize available Agilio software features and not interested in building their own datapath applications. The low level user datapath programming model is designed for users to program and customize the datapath on the SmartNIC. In this model, user programs can be written in pure P4, pure C, or a combination of both. For example, the flow table definition can be a P4 program, while the look-up actions are defined in a C program. We use the Netronome's SDK in our approach, with an integrated development environment that supports both P4 and C software development and debugging environment.

SmartNIC and Host-based NFV: As shown in Figure 3, packets arriving on the physical SmartNIC port (also called a physical function or PF) are processed by packet processing cores running either C or P4 functions. These functions can then direct packets to one or more Single-root input/output virtualization (SR-IOV) virtual function (VF) ports, which are exposed to the host. When combining a SmartNIC with a traditional NFV platform, packets will first be processed by the SmartNIC, then DMA'd to the host which will retrieve packets from the SR-IOV ports, and then the packets will be returned to the SmartNIC again, potentially for further processing or to be sent out to the physical port.

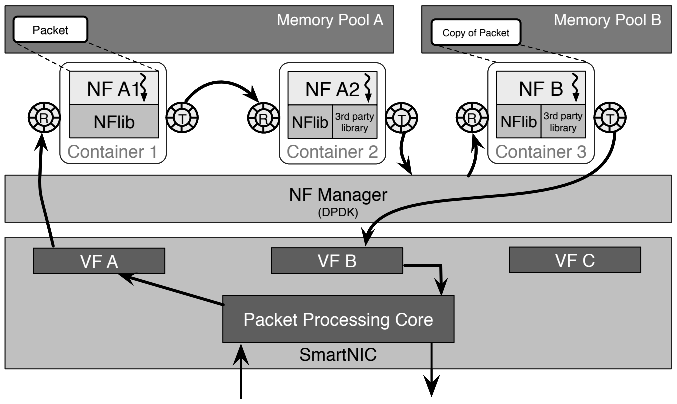

Fig. 3. System Architecture - Memory Isolation and SmartNIC

III. Advancing OpenNetVM with SmartNIC

We advance the OpenNetVM platform by leveraging the SmartNIC capabilities to improve the security and performance aspects.

A. Memory Isolation without SmartNIC

The current OpenNetVM platform has one global memory pool to keep the DMA'd packets. While this significantly improves the performance, there are security concerns as the memory pool can be read by all network functions. As a result, the trusted computing base (TCB) of the platform must include not only the manager, but the NFs as well.

We first extend the OpenNetVM platform by dividing the memory into separate regions based on the idea of NF tenants, and a service chain defines a set of functions to process packets across one or more tenants. A tenant might represent different network operators or virtual network customers. We assume that service chain classification can be performed by observing data within the packet header, e.g., based on the source/destination IP addresses. Instead of sharing the global memory pool, the NFs owned by a tenant can only access their own private memory region. While a packet is being transferred, the NF notifies the manager about the destination NF. If that is owned by a different tenant, the manager will copy the packet to another memory pool.

With this extension, the TCB of the OpenNetVM now excludes the network functions. That is, the tenants are isolated from each other and cannot eavesdrop on packets not destined for them. When the NF manager initializes, it creates multiple memory pools with unique pool IDs. If a service chain is entirely within a single tenant, then copies are not required.

However, this approach is problematic if packets arriving at the host are destined for different NFs at the *start* of their service chain. Since each tenant has its own memory pool, packets need to be loaded into the appropriate tenant pool before processing, but that cannot be determined until after the packet has already been DMA'd into the host's memory. To resolve this, we must use a simple Router function inside the NF Manager. This is a trusted component that retrieves incoming packets from the NIC and copies them into the pool of the appropriate tenant, resolving the trust issue, but negatively affecting performance since an additional copy is required for all packets. Further, the router function requires an additional CPU core that polls for packets, reducing the efficiency of the overall system.

B. SmartNIC-based Tenant Classification

We next explore how to improve our approach by offloading the tenant classification from a router NF on the CPU to the SmartNIC. Determining the service chain and tenant associated with an incoming packet simply requires matching a portion of the header to a rule set and then forwarding the packet appropriately, making this function a good fit for a P4 program. A flow table is preconfigured with the match rules,

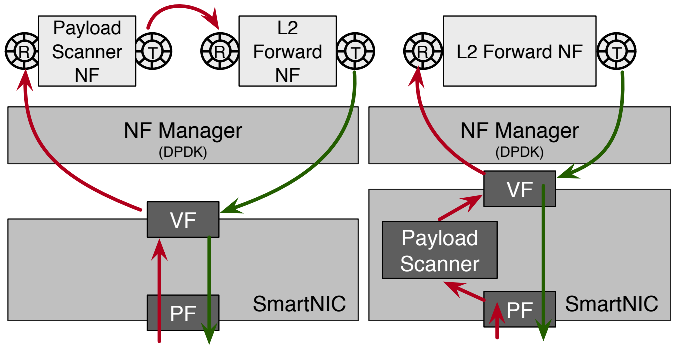

Fig. 4. Payload Scans in Smart NIC and CPU

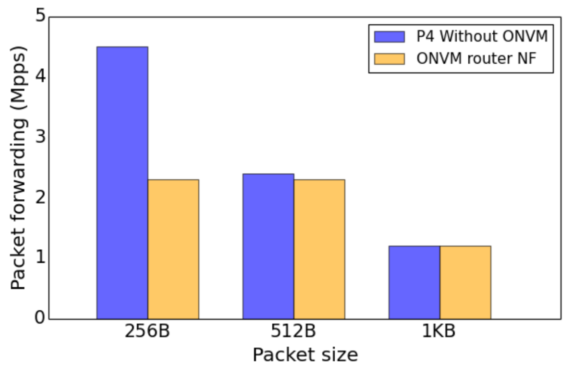

Fig. 5. Throughput with different packet forwarding techniques

and packets are sent out to different SR-IOV virtual functions (VFs). On the host, each VF is associated with a different memory pool, ensuring isolation.

Figure 3 shows an example of how the system works in a single round. When a packet arrives at the SmartNIC, it will first get analyzed by a packet processing core. The program accesses the packet header for essential information, for example, source/destination IP address, and looks up the pre-defined flow table. If there is a match on tenant ID ”A”, the packet will be forwarded to the virtual port A. The VF A is associated with memory pool A, which can only be accessed by NF A1 and NF A2. The manager notifies NF A1 to pick up the packet, process it, and send it to NF A2. Since the two NFs are from the same tenant, the manager allows the fast direct transmission. NF A2 processes the packet and tries to send it to NF B. Because the two NFs belong to different tenant, the manager copies the packet from pool A to pool B and notifies NF B. In the last step, the packet is sent out of the system from VF B, because it is associated with memory pool B. By leveraging the SmartNIC, the extra initial copy has been avoided, and for the common case where all NFs in a service chain are owned by a single tenant, full zero-copy IO is achieved while retaining tenant isolation with a lower TCB and more efficient resource utilization.

C. *SmartNIC Function Offload*

The SmartNIC’s flexibility and programmability means that it can do more than simple header matching as used in the prior section. Next we explore how deploying more complex network functions to the SmartNIC can potentially reduce both latency and CPU core usage.

While the SmartNIC Network Flow Processor (NFP) supports P4 programming for network control plane functionalities, this can also be combined with C programs to trigger “primitive actions” based on a match in the flow table. In Figure 4, we take a payload scan function as an example, which searches for a target string inside each packet’s contents. This type of payload analysis is a poor match for P4 programs, which work best on packets with well defined header structures.

Offloading this functionality to the SmartNIC instead of an NF running on the CPU has several benefits. First, the high degree of parallelism in the SmartNIC allows it to perform this functionality with high throughput. Next, performing the work on the NIC frees up CPU cores for other purposes. Finally, in some cases it may be possible to fully process a packet on the NIC, which can substantially reduce latency by avoiding the DMA and CPU entirely.

IV. Evaluation

In this section we evaluate how the SmartNIC can accelerate the OpenNetVM platform. We evaluate the advances of the OpenNetVM platform using the SmartNIC with the following goals:

- Demonstrate how the memory isolation by leveraging the SmartNIC is achieved without sacrificing performance (§IV-A),
- Show the function offloading can perform better in the SmartNIC rather than NF functions in CPU (§IV-B).

In our experimental setup, we configure two HP ProLiant GL160 G6 servers with two Intel Xeon X5650 CPUs @ 2.67 GHz with Ubuntu 16.04, Linux kernel 4.4.0. Each has an Agilio CX 2x10GbE SmartNIC. We generate network traffic with the Intel 82599ES 10 Gigabit Ethernet Controller on one server, send to the other one, and bounce the packets back to the sender.

A. *Memory Isolation*

Different from the global memory pool in OpenNetVM where packets are DMA’d first, then routed using packet descriptors, the memory isolation requires a routing function to steer packets into the right memory region. We compare the two scenarios when the function is in the CPU as a router NF vs. when in the SmartNIC. The P4 program first sees the packet header tuples, and based on the flow table entries created by SDN or a configuration file, it forwards (DMAs) the packets into the right memory region. The dedicated flow processing cores run this P4 program. On the other hand, in the CPU routing, the router NF receives a packet first, then forwards (copies) it to the correct memory pool for the next NF to process. Here we are using a hard-coded service chain rule for our evaluation. In the real world, either the router

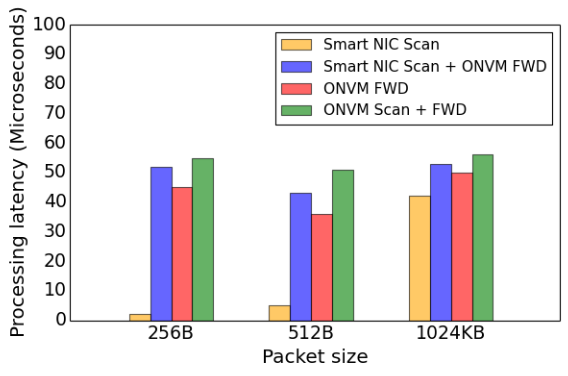

Fig. 6. Latency comparison of payload scanning in Smart NIC and CPU

NF or smart NIC will be able to communicate with the SDN controller to know the service chain rule, and find the next pool for the packet transmission.

Figure 5 shows the throughput comparison between P4 based packet routing and CPU based packet routing. For small packets, P4 routing achieves the line rates (i.e. 10GB), whereas the CPU routing suffers performance drops because the router NF running in CPU overloads the CPU cycles and packets are dropped. For larger packets, the overhead on the CPU is amortized since there are fewer packets per second, however, for network rates of of 40Gbps and beyond, we expect to see even more benefits for the SmartNIC.

B. *Function Offloading*

Next we demonstrate how the payload scanning function can perform better in the SmartNIC rather than in the NF running on CPU. In the SmartNIC Scan approach, the payload scan is entirely on the NIC before being sent out, with no CPU intervention. Figure 6 shows the processing latency (i.e., not including the base network RTT between the two servers) for different approaches to payload scanning. The yellow bar demonstrates the processing latency while the SmartNIC does the payload scan, and send the packets out directly. The green and blue bars show how OpenNetVM framework performs the same workload comparing to offloading the payload scan function to the SmartNIC. While the SmartNIC offloading (Smart NIC Scan + ONVM FWD) is able to achieve the full line rate throughput as well as the non-offloading framework, it shows up to 8 μs lower processing latency compared to the pure-software OpenNetVM approach (ONVM Scan + FWD). This is because the large number of flow processing cores efficiently processes the packets in parallel. Our evaluation result shows that the SmartNIC itself has much higher efficiency to process the networking workloads rather than CPU cores. SmartNIC offloading technique can be integrated into traditional NFV platforms, and help cloud developers achieve high performance, low latency, and save the CPU resources.

V. Related Work

Performance vs. Security: Previous NFV platforms [3], [8], [11], [13], [17] have focused on removing the processing overheads in the datapath to improve the performance, and use techniques such as zero-copy I/O and kernel bypass. Although these platforms can achieve high throughput, they also break the isolation between NFs and bring security concerns for the multitenant environment. Netbricks [12] proposed to use safe languages and runtimes to provide isolation with low overhead, but it requires rewriting all the NFs. IOVTee [14] added more control on top of NetVM to safely expose the host memory to the NFs, but it still suffers the same problem for the communication between NFs. Our work strengthens OpenNetVM by replacing the single mempool with multiple mempools for different tenants, and copying packets for across pool communication to provide isolation.

Hardware Acceleration: There have been many efforts to use inexpensive and programmable hardware such as FPGA, GPU, and Programmable NICs, to boost the performance of network applications. APUNet [6] demonstrated the effectiveness of using GPU to accelerate packet processing especially for compute-intensive network applications. G-Net [16] proposed a CPU-GPU NFV system to efficiently share GPU among multiple NFs. ClickNP [9] showed how to achieve both high performance and programmability on top of FPGA for general network functions. AccelNet [5] built and deployed FPGA-based programmable NICs in Azure data centers to offload host networking stacks to the hardware. Our work effectively integrates SmartNIC into OpenNetVM to offload partial network processing with the goal to balance the performance and isolation.

VI. Conclusion and Future Work

Existing Network Function Virtualization platforms have achieved high throughput and low latency by using dedicated CPU cores and zero-copy techniques, incurring high resource consumption and reduced isolation. In this paper, we have explored how programmable NICs can be used to reduce CPU load and improve isolation. We have integrated a programmable NIC into OpenNetVM NFV framework which can offload the computation from CPU and route packets to isolated memory regions. The results show it can improve the throughput by 2 million packets per second and decrease the latency by at most 8 μs.

Our vision is to offload all NFV platform functionalities and standard NF functions to the SmartNIC so that the CPU can only run the application logic. The first issue is to make the zero-copy to completion. Currently, we use the conditional packet copy on the cross-pool transmission. Although we leverage the SmartNIC to improve the packet forwarding, the overhead from the packet copy is significant as it proportionally increases depending on how many times that cross-pool transmission requests are made. We plan to offload all the packet transmissions to SmartNIC to eliminate the packet copy overhead. That is, all the transmission across different memory pools will go through the SmartNIC processor via DMA. In the end, the platform could achieve zero CPU copy I/O to completion.

Another issue to tackle is the SmartNIC virtualization. Hyper4 [7] has attempted to virtualize the SmartNIC using P4 only for the flow management, but it does not virtualize the function level including microC code. Since SmartNIC approach allows only one program to be loaded for all SR-IOV interfaces at the same time, there could be potential interference between different NFs that might render security concerns. As the SmartNIC hardware accelerators support the security of network application development, we plan to implement isolation programmatically on the hardware level.

Acknowledgements: This work was supported in part by NSF grants 1525992 and 1814234.

References

[1] Netronome - Smart NICs. https://www.netronome.com. [ONLINE].

[2] P4 language consortium. https://p4.org/. [ONLINE].

[3] Data plane development kit. http://dpdk.org/, 2019. [ONLINE].

[4] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., and Walker, D. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev. 44*, 3 (July 2014), 87–95.

[5] Firestone, D., Putnam, A., Mundkur, S., Chiou, D., Dabagh, A., Andrewartha, M., Angepat, H., Bhanu, V., Caulfield, A. M., Chung, E. S., Chandrappa, H. K., Chaturmohta, S., Humphrey, M., Lavier, J., Lam, N., Liu, F., Ovtcharov, K., Padhye, J., Popuri, G., Raindel, S., Sapre, T., Shaw, M., Silva, G., Sivakumar, M., Srivastava, N., Verma, A., Zuhair, Q., Bansal, D., Burger, D., Vaid, K., Maltz, D. A., and Greenberg, A. G. Azure accelerated networking: Smartnics in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018* (2018), pp. 51–66.

[6] Go, Y., Jamshed, M., Moon, Y., Hwang, C., and Park, K. Apunet: Revitalizing gpu as packet processing accelerator. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2017), NSDI'17, USENIX Association, pp. 83–96.

[7] Hancock, D., and van der Merwe, J. Hyper4: Using p4 to virtualize the programmable data plane. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies* (New York, NY, USA, 2016), CoNEXT '16, ACM, pp. 35–49.

[8] Hwang, J., Ramakrishnan, K. K., and Wood, T. Netvm: High performance and flexible networking using virtualization on commodity platforms. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)* (Seattle, WA, 2014), USENIX Association, pp. 445–458.

[9] Li, B., Tan, K., Luo, L. L., Peng, Y., Luo, R., Xu, N., Xiong, Y., Cheng, P., and Chen, E. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference* (New York, NY, USA, 2016), SIGCOMM '16, ACM, pp. 1–14.

[10] Liu, J., Hallahan, W., Schlesinger, C., Sharif, M., Lee, J., Soulé, R., Wang, H., Caşcaval, C., McKeown, N., and Foster, N. P4v: Practical verification for programmable data planes. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (New York, NY, USA, 2018), SIGCOMM '18, ACM, pp. 490–503.

[11] Palkar, S., Lan, C., Han, S., Jang, K., Panda, A., Ratnasamy, S., Rizzo, L., and Shenker, S. E2: A framework for nfv applications. In *Proceedings of the 25th Symposium on Operating Systems Principles* (New York, NY, USA, 2015), SOSP '15, ACM, pp. 121–136.

[12] Panda, A., Han, S., Jang, K., Walls, M., Ratnasamy, S., and Shenker, S. Netbricks: Taking the v out of nfv. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2016), OSDI'16, USENIX Association, pp. 203–216.

[13] Rizzo, L. netmap: A Novel Framework for Fast Packet I/O. In *USENIX Annual Technical Conference* (Berkeley, CA, 2012), USENIX.

[14] Ryota Kawashima, H. M. Iovtee: A fast and pragmatic software-based zero-copy/pass-through mechanism for nfv-nodes. In *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Verona, Italy, November 27-29, 2018* (2018).

[15] Stephens, B., Akella, A., and Swift, M. M. Your programmable nic should be a programmable switch. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks* (New York, NY, USA, 2018), HotNets '18, ACM, pp. 36–42.

[16] Zhang, K., He, B., Hu, J., Wang, Z., Hua, B., Meng, J., and Yang, L. G-NET: effective GPU sharing in NFV systems. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018* (2018), pp. 187–200.

[17] Zhang, W., Liu, G., Zhang, W., Shah, N., Lopreiato, P., Todeschi, G., Ramakrishnan, K., and Wood, T. OpenNetVM: A platform for high performance network service chains. In *HotMiddlebox* (2016), ACM.