Network Programming Interface in General-Purpose Multi-core Processor: A Survey

Jinli Yan, Chunbo Jia, Lu Tang*, Tao Li, Gaofeng Lv, Wei Quan and Hui Yang
College of Computer, National University of Defense Technology
Email: {yanjinli10, jiachunbo, lutang, taoli, lvever, w.quan, yanghui}@nudt.edu.cn

***Abstract*—Recently, the development and deployment of dedicated hardware can hardly keep pace with the increasing complexity of the network functions. With the performance improvement, the maturity of virtualization technology and many open-source network projects (such as DPDK), the general-purpose multi-core processors (multi-core CPU) have been widely used in network field. Network operators are migrating network functions to multi-core CPU for flexible development and dynamic deployment. Network programming interfaces on multi-core CPU not only provide general packet processing services but also shield the complex implementation of software and hardware for developers. Both network application requirements and implementation mechanisms of the underlying platform have significant influences on the development of network programming interfaces. In this paper, we first review the development of network programming interfaces and classify programming interfaces by information model, resource management mode, and development model. We then analyze the programming interfaces provided by Click, VPP, and mOS, respectively. Because 5G and data center networks require high programmability and performance, and software/hardware co-processing become popular packet processing architectures, we also propose future research directions, including application-defined semantics of control blocks, software/hardware dynamical reconfiguration, and multi-dimensional event management.**

***Index Terms*—General-Purpose Multi-core Processor, Packet Processing, Network Programming Interface**

I. Introduction

Currently, the commercial ASIC-based network devices are designed for providing dedicated network functions, such as routing and switching. Since the providers only open limited configuration interfaces, many end-to-end communication functions are programmed on the generic end systems. As the network size and traffic increase, network devices are required to support rapid development and dynamic deployment of new network functions for meeting the requirements on network security, reliability, and quality of service (QoS) [6]. Since customizing a dedicated network device takes long development cycles and high costs, the traditional method impedes the innovation and application of network technologies.

The existing platforms on which users develop applications are divided into three types. (1) Network Processor (NP) [12]. NP uses a multi-core parallel processing architecture and is dedicated for packet processing. Although the processing performance of NP is very high, it is difficult to develop applications with a dedicated instruction set. (2) Programmable switching chips. A typical representation is RMT architecture that provides multi-level "Match→Action" [17]. The compiler compiles the code written with domain-specific high-level language named P4 [16] into the underlying hardware logic. However, the programmable switch chips are difficult to support complex processing logic, such as stateful processing. And their performance mainly depends on the optimization of the compiler. (3) General-Purpose Processor (GPP, also known as CPU). At present, the CPUs adpot multi-core processing architecture and the underlying differences are shielded by the operating system. Thus, diversified high-level programming languages are provided for users to develop.

In this paper, we focus on researching the network programming interfaces based on multi-core CPU. In addition to end systems, the multi-core CPUs are also widely applied in network access devices and network middleboxes, including vBRAS [11], firewall, intrusion detection system. The main reasons why multi-core processors are gradually replacing dedicated network equipment in the network edge are summarized as follows. (1) Since the functions deployed on the network edge devices become more complex and diverse, the programmability is required to improve. The software programming on CPU has flexibility and users develop applications with high-level languages (such as C/C++) without caring about the differences of underlying hardware. (2) The processing performance of CPU improved greatly when it is developed from single-core to multi-core architecture. Meanwhile, the technology innovations in PCIe and DDR promote the improvements in memory access bandwidth and peripheral transmission bandwidth. With these hardware technologies, CPUs could support high bandwidth and complex network processing better. (3) The maturity of virtualization technology promotes the development and deployment of network functions on virtual machines and containers, which accelerates the delivery of network functions [33]. (4) The development of the open-source ecological environment in the network field provides good support for network innovations. We take the high-performance data plane development kit (DPDK) [5] as an example. With the optimizations of DPDK, including polling mode, userspace driver and core affinity, the throughput of software processing on CPU could reach up to 10Gbps, even 100Gbps.

Importantly, the network programming interfaces on the CPU plays a key role in the development of network functions. As the bond between the underlying platform and users,

*Lu Tang is the corresponding author.

network programming interfaces provide users with a series of general packet processing services and development specifications. At the same time, it shields the complex implementation details of the underlying software and hardware and reflects the design model and implementation mechanism of platforms. Therefore, the design of network programming interfaces is determined by the upper application requirements and the implementations of the underlying platform. There are two key factors affecting the design of programming interfaces. (1) The function division between application and platform. The interface design is affected by the function division. For example, L2-L4 processing is performed in the operating system and the upper applications process the application data directly. Since DPDK does not provide a protocol stack, the L2-L4 protocol parse is implemented based on the integrated packet data. (2) Resource view of applications provided by the platform. The type and number of resources affect the design of the programming interfaces. For instance, the OS provides thread and buffer allocation and de-allocation interfaces for users to manage the resources. The vector packet processor (VPP) provides packet vectors for users without exposing the buffer and thread resources [14].

Although the network subsystem in the operating system provides system calls for packet receiving and sending, these programming interfaces are difficult to meet the requirements of developing high-performance network functions. The main reason is that multi-core CPUs are designed for supporting computing, storage, and networking applications, and the design of software and hardware is not customized by considering the features of network processing. Recently, there are plenty of researches focusing on optimizing the performance and programming efficiency of network applications on multi-core CPUs and enriching the network programming interfaces. With respect to performance, the related researches, such as netmap [37], PF_RING [38], DPDK [5], VPP [14], Packetshader [22] and SwitchBlade [13], focus on the packet IO and packet processing optimization. As for programming efficiency, the optimizations in typical works, including Click [27], ODP (Open Data Plane) [7], mOS [25], Microbox [32], concentrate on the modular development, application portability, and platformization of common functions.

Although there are some review articles making analyses and comparisons on the underlying implementation principles of network programming interfaces, comparing and summarizing these interfaces from the user perspective is scarce and urgently needed. In this paper, we first classify and compares many typical programming interfaces from three dimensions, including information model, resource management mode and development model. Based on these analyses, we propose that application-defined semantics of control block, dynamic deployment of network functions and multi-dimensional event management are future research directions. This work would help developers design user-friendly programming interfaces and improve the development efficiency of network functions.

The rest of the paper is organized as follows. We introduce the background of network programming interfaces in Section II. We make a detailed classification and comparison of programming interfaces in Section III. We discuss the future directions in Section IV and conclude the paper in Section V.

II. Background

Since TCP/IP protocol was proposed, academia and industry have proposed plenty of network processing technologies to optimize and improve the performance and programming efficiency of the network processing platform. These network processing technologies provide programming interfaces for users to develop and deploy network functions. Therefore, the research on network programming interfaces should start from the related technologies. In this section, we review the development process in terms of application requirements and platform implementation. Since we focus on analyzing the programming interfaces from the user perspective, the design ideas of the underlying platform are only sketched in this paper.

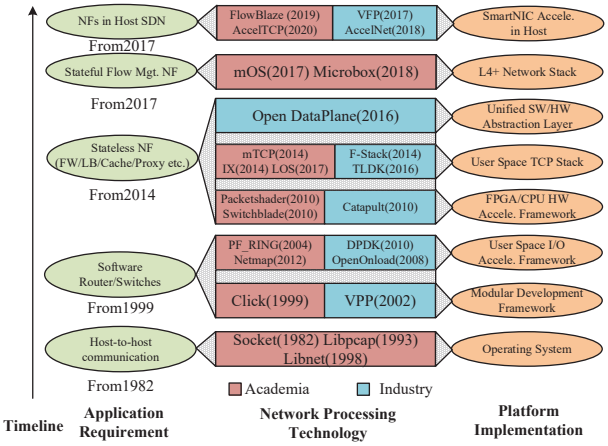

Fig. 1. Development of Network Packet Processing Technologies.

We summarize the development of network processing technology in academia and industry, as shown in Fig. 1. Before 1999, network equipment customized with dedicated chips only provide users with limited configuration interfaces with low programmability. There are some basic communication functions developed on the end systems with the network-related system calls. The representatives in this phase include Socket [18], Libpcap [24], Libnet [39], etc. After 1999, with the enrichment of network protocols and policies, the routing and switching functions began to migrate from dedicated ASIC chips to general-purpose processors. It enables the rapid development and deployment of software functions at the network edge. The platform implementation at this stage mainly includes three features. (1) Modular design. The typical representatives contain the Click [27] software routers proposed by MIT in 1999 and VPP [14] [1] designed by Cisco in 2002. (2) Improving IO performance. With the performance improvement brought from multi-core architecture, the network IO performance does not adapt to computing performance very well. Developers design and implement user-space IO acceleration framework, including netmap [37],

[1]It becomes an open-sourced subproject of FD.io community [4] in 2016.

PF_RING [38] in academia, and OpenOnload [9] and DPDK [5] in industry.

In 2002, Network Function Virtualization (NFV) [6] proposed to decouple the software from the general-purpose multi-core processors. The traditional customized middleboxes gradually migrated to multi-core CPUs, including firewalls, IDS, cache, proxy, and other stateless network functions. The researches about platform implementation contain two points. (1) Hardware acceleration framework. The architecture of network devices began to change from the CPU-based homogeneous model to a heterogeneous model that integrates CPU and hardware accelerators. The typically applied hardware accelerators include FPGA and GPU. The research work in academia includes PacketShader [22], SwitchBlade [13], ClickNP [29] and APUNet [21]. For example, PacketShader proposed to build a high-performance software router based on CPU/GPU architecture. SwitchBlade accelerates the deployment of customized protocols on CPU/FPGA platform. In industry, Microsoft's Catapult project [1] focuses on accelerating the cloud computing platform with a large-scale deployment of FPGAs in data centers. (2) User-space protocol stack. Although netmap, PF_RING, and DPDK improve the IO performance greatly by bypassing the kernel, the user-space TCP/IP protocol stack is scarce at that time. The user-space protocol stack is urgently needed for reducing the development efforts of complex network functions. There are some well-known achievements, such as the mTCP [26], IX [15] and LOS [23], F-Stack [2] of Tencent and TLDK [10] in FD.io.

With the diversified development of network programming frameworks and hardware platform architectures, the differences in programming interfaces from different platforms increases. Since the cross-platform migration of applications is tedious and time-consuming, designing platform-independent generic interfaces is essential for supporting the seamless migration. The Open Data Plane project [7], launched in 2016, provides unified programming framework by designing an abstraction layer to shield the difference of system calls and programming interfaces in other platforms. DPDK provides a device abstraction layer for adapting to different hardware accelerators. It makes the underlying hardware transparent to users.

With the more sophisticated requirements on network security and QoS, the research about network functions is changing from L2-L3 to L4-L7. The processing logic of network functions becomes more complex so that the difficulty and workload of development have further increased. Recently, there are plentiful works on stateful flow management applications, including stateful firewall, Layer 7 load balancing, etc. Although these network functions deliver different services, there is much overlapped logic in them. Therefore, designers migrate the common functions from the user application logic to the underlying platform and provide users with L4+ protocol stack. For example, mOS [25] and Microbox [32] support the rapid development of L4-L7 stateful network functions with a stateful network stack in userspace.

With the development of 5G and data center networks, Software-Defined Networking (SDN) [28] facilitates the end-to-end deployment of network functions for improving resource utilizaton and reducing operating costs. The applications of Host SDN include ACL, Load Balancer, and vNet. There are some preliminary works, including FlowBlaze [36] and AccelTCP [34] in academia, Microsoft's VFP (Virtual Filtering Platform) [19], AccelNet (Azure Accelerated Networking) [20] and open-source project OpenNFP [8]. FlowBlaze and AccelTCP focus on offloading the stateful packet processing and TCP stack into programmable SmartNIC respectively. VFP architecture implements software functions in a virtual switch and accelerates these functions with programmable SmartNIC. AccelNet proposes to offload the network stack of host SDN into SmartNIC. The fast path of Open vSwitch (OVS) [35] is offloaded into SmartNIC in OpenNFP.

III. Classification and comparison

From the analysis about the development process of network technologies, there are three factors promoting the development and evolution of programming interfaces. (1) The high-tech chips in network equipment are experiencing the developing trends of dedicated ASIC chips $\rightarrow$ multi-core CPU $\rightarrow$ multi-core CPU/hardware coprocessors. Since multi-core CPUs are adopted in network devices to improve programmability, diversified programming interfaces are designed and applied recently. (2) The development of packet IO and processing technology. These performance optimization technologies propose high-efficiency processing mechanisms by improving OS kernel, bringing differences in the design of programming interfaces. (3) Network functions become complicated. Except for the switching and routing functions in L2-L3, there are many virtualization-related protocols and stateful processing functions in L4-L7. Thus, the programming interfaces are enriched because the information to be processed increases and the processing logic becomes more complicated.

Therefore, network programming interfaces show a diversified development trend in recent years. Apart from improving packet IO and packet processing performance, these programming interfaces also focus on improving the developing efficiency by shielding the complex underlying implementations. In this paper, we select some researches that propose typical network programming interfaces in academia and industry, including Socket, Click, VPP, netmap, PF_RING, DPDK, mTCP, ClickNP, ODP, mOS and Microbox. These network programming interfaces are classified and compared from three dimensions: information model, resource management mode and development model.

A. Classification based on Information Model

The information model covers the data objects and their relationships between the underlying platform and applications. It is independent of the specific data transmission protocol and implementations. The design of programming interfaces must consider the information provided for developers firstly. The data between platform and applications consists of packet data and metadata. Packet data is the packet information

received by the network devices, including the packet header and application-related data. Metadata refers to the control data that is generated and processed only in the network devices. It contains the data that is related to the packet parse, preprocessing, etc. In this paper, we classify the information model into three categories: raw packet data (RP), discrete packet data/metadata (D-PM), and integrated packet data/metadata (I-PM).

Raw packet data (RP). It is the complete or partial packet information provided by the platform for the application processing. The interfaces of Socket and mTCP provides the raw packet data. The Socket API includes three types. The socket in the transport layer only provides application-related messages. The sockets in the network layer and link layer offer the data including IP header and Ethernet header respectively. mTCP provides socket-like APIs that shield the processing of TCP stack and expose the application data directly.

Discrete packet data/metadata (D-PM). It means that the control info and packet data are stored in two sperate buffers. The typical representatives include netmap, PF_RING, Click, mOS, and Microbox. The metadata generally contains packet length, packet address, input port, timestamp, etc.

Integrated packet data/metadata (I-PM). It uses a large buffer to store the control info and packet info. I-PM mode is adopted by DPDK, VPP, ODP, and ClickNP. For example, there are plentiful features in the metadata of DPDK, including data block address, fragmentation information, input port, L2-L4 parse results, TX offload, etc. Moreover, DPDK and ODP reserve a headroom between the data block and control block and a tailroom at the end of the packet. Thus, developers could adjust the message space with the specified APIs flexibly.

B. Classification based on Resource Management Mode

The resource management mode means that the allocation/deallocation operations of resources (such as threads and memory) are performed on platforms or applications. Since the function partitioning between platform and application affects the interface design, here we divide the resource management mode into the explicit mode and implicit mode.

Explicit resource management. It requires users to call the thread and memory management interfaces. That means these resources are visible for users. With the increase of network bandwidth, the frequent buffer allocation/deallocation operations would generate substantial overhead. Thus, resource management is handed over to the platform, which is transparent to users. We classify these objects in terms of the thread management mode. The explicit thread management is used in Socket, netmap, PF_RING, DPDK, mTCP and ODP. In this mode, developers could customize resource management according to the application features. It is suitable for professionals in performance optimization. However, implementing complex resource management and application logic is difficult and takes great development efforts.

Implicit resource management. It means that the underlying platform executes the management and control of thread and memory resources. It only provides related configuration interfaces for developers. The network processing technologies that adopt implicit resource management include Click, VPP, ClickNP, mOS, and Microbox. In this mode, the platform performs the packet IO and processing operations by starting threads actively. Users could configure the number of threads and set the CPU affinity. Therefore, developers focus on the development of application logic. The disadvantages are that managing resources on the underlying platform may not adapt to the requirements of performance and resource utilization in specified scenarios.

C. Classification based on Development Model

Currently, the modular design architecture that many network systems adopt has three advantages: (1) dividing the complex logic into multiple modules makes collaborative development much easier. (2) the difficulty of debugging and verification is decreased at the module level. (3) the modules could be loaded or unloaded without programming the overall logic from scratch. The modular architecture has an important impact on the development model according to the specific module abstraction and division. Here we divide programming interfaces into packet-centric and event-centric from the perspective of the development model.

Packet-centric development model. It transforms network function logic into a directed graph where each node and each edge represents a module and connections respectively. The typical works include Click, VPP, ClickNP, and DPDK. Click and ClickNP configure the module connections with an extra configuration script. Differently, VPP and DPDK achieve the connections by calling specific programming interfaces. The interfaces for each module could be abstracted into the local state and functions. The functions are responsible for the initialization and configuration.

Event-centric development model. It classifies the data to be processed into various events. The programming interfaces of each module are abstracted into events and event processing functions. This method is applied in Socket, mTCP, netmap, PF_RING, ODP, mOS, and Microbox. The Socket, mTCP, netmap, and PF_RING provides IO multiplexing programming interfaces named epoll [31]. Epoll only provides coarse-grained event abstractions, including message reception, message transmission, and message error. ODP writes the event type into the control block and classifies the events into different queues for further processing. mOS and Microbox provide the events related to packet data and flow states. Besides, users could customize new event types and event filtering functions according to the application requirements.

D. Comprehensive comparison

We classify these programming interfaces mentioned above into different areas, as shown in Fig. 2. Here we make a comprehensive comparison of these programming interfaces from three aspects and summarize the future development of programming interfaces.

Information Model. Computing and storage applications mainly process the data in the application layer without caring

about the packet header and metadata. On the contrary, most network functions focus on processing the packet header and rich control info for security analysis and QoS scheduling. Besides, in order to offload some network functions, the underlying hardware is required to provide abundant intermediate processing results.

Resource Management Mode. The packet IO frameworks provide related interfaces to manage thread resources for professional developers. With the development and application of network processing frameworks, such as VPP and Click, these resources are managed by platform and users only need to design and implement the application logic. At present, the automated operations and DevOps technology receive much attention for reducing the development efforts. The platform would provide more general services so that the implicit resource management would gradually be widely applied.

Development Model. The programming framework for general network processing mainly adopts packet-centric development. However, for the complex functions, the disadvantages of this method are the low module reuse rate and development efficiency. To solve this problem, the application-specific programming frameworks adopt event-centric development. The platform provides a fine-grained event view for upper users to register various functions.

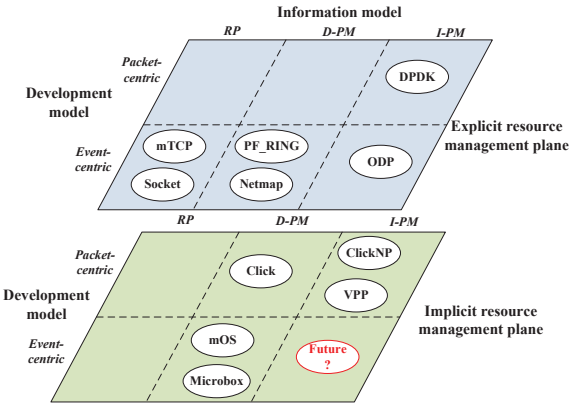

Fig. 2. A Comprehensive comparison of Network Programming Interfaces.

From the overall development trends, the current research hotpots on network processing technologies have changed from network IO to network processing. In order to reduce the development efforts and focus on the network processing logic, resource management is gradually taken over by platforms. In order to provide more sophisticated services, such as load balancing and security monitoring, more control info is required to expose to users. Since I-PM has higher management efficiency than D-PM, it is a more suitable selection of interface design in the future. As more programming frameworks are designed for dedicated application scenarios, the event-centric development has advantages on improving efficiency and flexibility. In summary, the features of future programming interfaces are the integrated packet data/metadata information model, the implicit resource management mode, and the event-centric development model.

IV. Future Directions

With the rapid development of 5G network and data center networks, improving the network programmability and processing performance becomes the future development trend. Since the software and hardware have advantages in programmability and performance respectively, it is important to make a good tradeoff. Thus, the software/hardware heterogeneous processing architecture abstracts widespread attention. At present, there are some existing works [40] [30] and open-source projects [3] on accelerating the virtual switches and middleboxes of NFV with FPGA. And some prototypes (such as SmartNIC) have been applied. Thus, it is important to design programming interfaces based on the heterogeneous architectures. In this paper, we propose three directions to provide references for researches.

Application-defined semantics of control block. The control block is an important data structure for the coprocessing between software and hardware in heterogeneous architecture. By providing rich control info for upper applications (such as counters, network load), the network security and reliability could be further improved. Meanwhile, the hardware could complete the post-processing and function accelerations with the intermediate results in the control block. The current challenge is that there is a difference among the control info for various applications. However, designing a large and comprehensive control block would generate extra communication overhead because lots of control info is useless. Thus, the semantics of the control block should be defined by applications including the content and structure of control info.

Dynamic deployment of software and hardware functions. With the continuous update and change of requirements, network devices need to support the dynamic deployment of network functions. In heterogeneous architecture, the network functions are separated into software functions and hardware functions. In the open-source mode, developers implement and submit the function modules to the project. It builds a rich module library for improving development efficiency. The main challenge is to transform from the code-level development to module-level deployment. Thus, the programming interfaces should provide orchestration services. The platform would search and load the required function modules from the library dynamically without interrupting the normal system execution.

Multi-dimensional event management. At present, the network events are changing from the low-level events that only have packet data to the high-level events including flow states. In heterogeneous architecture, the system events would be enriched further, like the state synchronization events. The classification and management of events decide how to decouple the network functions. If the semantics of some events are overlapped, there are some repetitive processing logic in the related modules. Moreover, the granularity of events that different applications process is different. Some applications would process some aggregated events. Thus, it

is important to classify the events from multiple dimensions (such as the application and platform features) for designing a high-efficiency event management mechanism.

V. Conclusion

This paper is a survey about network programming interfaces of general-purpose multi-core processors. To implement the flexible development and rapid deployment of network functions, more and more network functions are gradually implemented based on multi-core CPUs. Recently, many research works about network processing technologies enriched the programming interfaces that play an important role in determining the processing performance and development efficiency. In this paper, we first summarize the development process of network processing technologies in terms of application requirements and platform implementations. Then many typical programming interfaces are classified and compared from the perspective of information model, resource management mode and development model. Finally, we propose important research directions as a reference for the innovations of network technologies and programming interfaces.

Acknowledgment

This work is supported by National Key Research and Development Program of China (Grant No.2018YFB1800505, 2018YFB1800402), National Natural Science Foundation of China (Grant No.61702538, 61802417, 61601483), Training Program for Excellent Young Innovators of Changsha (Grant No.kq1905006) and Research Project of National University of Defense Technology (Grant No.ZK17-03-53, ZK18-03-40).

References

[1] Catapult Project. https://www.microsoft.com/en-us/research/project/project-catapult.

[2] F-Stack. http://www.f-stack.org.

[3] FAST Project. http://www.fastswitch.org.

[4] FD.io: Fast Data Project. https://fd.io.

[5] Intel Data Plane Development Kit. https://www.dpdk.org/.

[6] Network Functions Virtualization Introductory White Paper. http://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[7] Open Data Plane Project. https://www.opendataplane.org.

[8] OpenNFP. https://open-nfp.org.

[9] OpenOnload. https://www.openonload.org.

[10] TLDK. https://wiki.fd.io/view/TLDK.

[11] vBRAS: Virtualized Broadband Remote Access Server. https://marketplace.vmware.com/vsx/solutions/zte-vbras?ref=related.

[12] James R Allen, Brian M Bass, et al. IBM PowerNP Network Processor: Hardware, Software, and Applications. *IBM Journal of research and development*, 47(2.3):177–193, 2003.

[13] Muhammad Bilal Anwer, Murtaza Motiwala, Mukarram bin Tariq, and Nick Feamster. Switchblade: A Platform for Rapid Deployment of Network Protocols on Programmable Hardware. In *Proceedings of the ACM SIGCOMM 2010 conference*, pages 183–194, 2010.

[14] David Barach, Leonardo Linguaglossa, et al. High-Speed Software Data Plane via Vectorized Packet Processing. *IEEE Communications Magazine*, 56(12):97–103, 2018.

[15] Adam Belay, George Prekas, et al. IX: A Protected Dataplane Operating System for High Throughput and Low Latency. In *11th USENIX Symposium on Operating Systems Design and Implementation*, pages 49–65, 2014.

[16] Pat Bosshart, Dan Daly, et al. P4: Programming Protocol-Independent Packet Processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[17] Pat Bosshart, Glen Gibb, et al. Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN. *ACM SIGCOMM Computer Communication Review*, 43(4):99–110, 2013.

[18] Bill Fenner and Andrew M Rudoff. *UNIX Network Programming: The Sockets Networking API*. Addison-Wesley, 2004.

[19] Daniel Firestone. VFP: A Virtual Switch Platform for Host SDN in the Public Cloud. In *14th USENIX Symposium on Networked Systems Design and Implementation*, pages 315–328, 2017.

[20] Daniel Firestone, Andrew Putnam, et al. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation*, pages 51–66, 2018.

[21] Younghwan Go, Muhammad Asim Jamshed, et al. APUNet: Revitalizing GPU as Packet Processing Accelerator. In *14th USENIX Symposium on Networked Systems Design and Implementation*, pages 83–96, 2017.

[22] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. PacketShader: a GPU-Accelerated Software Router. *ACM SIGCOMM Computer Communication Review*, 40(4):195–206, 2010.

[23] Yukai Huang, Jinkun Geng, et al. Los: A High Performance and Compatible User-Level Network Operating System. In *Proceedings of the First Asia-Pacific Workshop on Networking*, pages 50–56, 2017.

[24] V Jacobson, C Leres, and S McCanne. Libpcap, Lawrence Berkeley Laboratory, Berkeley, CA. *Initial public release June*, 1994.

[25] Muhammad Asim Jamshed, YoungGyoun Moon, et al. mOS: A Reusable Networking Stack for Flow Monitoring Middleboxes. In *14th USENIX Symposium on Networked Systems Design and Implementation*, pages 113–129, 2017.

[26] EunYoung Jeong, Shinae Wood, et al. mTCP: a Highly Scalable User-Level TCP Stack for Multicore Systems. In *11th USENIX Symposium on Networked Systems Design and Implementation)*, pages 489–502, 2014.

[27] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. The Click Modular Router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[28] Diego Kreutz, Fernando MV Ramos, et al. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1):14–76, 2014.

[29] Bojie Li, Kun Tan, et al. ClickNP: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 1–14, 2016.

[30] J Li, X Yang, and Z Sun. Drawerpipe: Drawerpipe: A Reconfigurable Packet Processing Pipeline for FPGA. In *Journal of Computer Research and Development*, 2018.

[31] Davide Libenzi. Linux Epoll Patch, 2006.

[32] Guyue Liu, Yuxin Ren, Mykola Yurchenko, KK Ramakrishnan, and Timothy Wood. Microboxes: High Performance NFV with Customizable, Asynchronous TCP Stacks and Dynamic Subscriptions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 504–517, 2018.

[33] Rashid Mijumbi, Joan Serrat, et al. Network Function Virtualization: State-of-the-art and Research Challenges. *IEEE Communications surveys & tutorials*, 18(1):236–262, 2015.

[34] YoungGyoun Moon, SeungEon Lee, et al. Acceltcp: Accelerating network applications with stateful {TCP} offloading. In *17th USENIX Symposium on Networked Systems Design and Implementation*, pages 77–92, 2020.

[35] Ben Pfaff, Justin Pettit, et al. The Design and Implementation of Open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation*, pages 117–130, 2015.

[36] Salvatore Pontarelli, Roberto Bifulco, et al. Flowblaze: Stateful Packet Processing in Hardware. In *16th USENIX Symposium on Networked Systems Design and Implementation*, pages 531–548, 2019.

[37] Luigi Rizzo. Netmap: a Novel Framework for Fast Packet I/O. In *21st USENIX Security Symposium*, pages 101–112, 2012.

[38] Luigi Rizzo, Luca Deri, and Alfredo Cardigliano. 10 Gbit/s Line Rate Packet Processing Using Commodity Hardware: Survey and New Proposals, 2012.

[39] Mike Schiffman. The Libnet Packet Construction Library. *The Million Packet March*, 2005.

[40] Jinli Yan, Tao Li, Sheng Wang, Gaofeng Lv, and Zhigang Sun. Demonstration of Path-Based Packet Batcher for Accelerating Vectorized Packet Processing. In *2018 15th Annual IEEE International Conference on Sensing, Communication, and Networking*, pages 1–3. IEEE, 2018.