NF-Switch: VNFs-enabled SDN Switches for High Performance Service Function Chaining

Cheng-Liang Hsieh and Ning Weng
Department of Electrical and Computer Engineering, Southern Illinois University Carbondale, USA
Email: hsieh@siu.edu, nweng@siu.edu

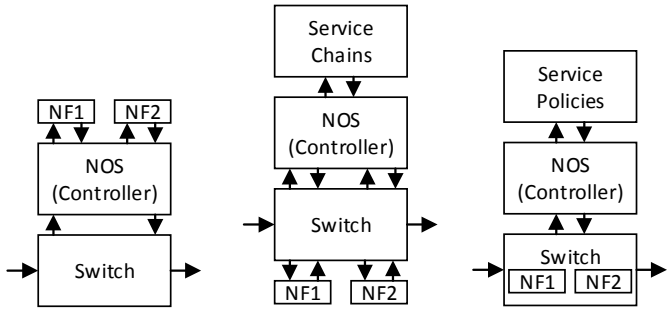
***Abstract*—Current Software-Defined Networking (SDN) switch examines many packet headers to support the flow-based packet forwarding. To support the application-based forwarding for a service function chaining (SFC), SDN switch requires packet header modifications to identify the processing status and multiple packet matchings to steer the network traffics to different VMs in a specific order. Both challenges require significant computation resources in a system and result in severe system performance degradation. To improve the system performance and keep the flexibility, this paper proposes NF-Switch to eliminate the requirement of packet modifications and to reduce the number of matchings for the application-based forwarding. Compared to the native implementation, our experimental results show that NF-Switch reduces the system processing latency to one quarter and increases the system throughput about 3 times for a SFC with 10 network functions (NFs). Moreover, the proposed solution maintains the content switching time to update the implemented SFC for a better system scalability regarding to the number of NFs in a SFC.**

I. Introduction

Software-Defined Networking (SDN) and Network Function Virtualization (NFV) create a network environment that promotes the innovations by removing the disadvantages of proprietary software and hardware. On the one hand, the programmable switches increase the network manageability at a low operation and management cost using the central control unit. On the other hand, NFV helps to improve the system flexibility without sacrificing the network functionalities [1] by leverage virtualization techniques. By integrating SDN with NFV, a service provider can find new use cases to satisfy the new demands of its users for better customer satisfactions and to generate higher revenues with a lower capital investment.

The design of a SDN switch is now simple since all control functions are running separately by another logical centralized network device. When a packet does not match any entry in the forwarding table, the switch passes this packet to the controller for the further operation. According to this basic forwarding behavior of a SDN switch, a more comprehensive controller like Network Operating System (NOS) [2] in SDN allows system administrators to add new network applications into an SDN network as modules of NOS as shown in Figure 1a. This setup generates a high system performance and maintains a good system flexibility for application like L2 forwarding when most packets go through the fast path. However, the system performance suffers when most packets go through the slow path for the required network functions like packet inspections.

978-1-5090-6501-1/17/$31.00



(a) NOS Apps (b) VNFs/Mibbleboxes (c) VNF-enabled SW

Fig. 1: An example of different network function implementations. The proposed switch design to equip NFVs as extension modules. The required VNFs are initiated based on the flow table matching results and are controlled by NOS.

To improve the system performance and the system flexibility, a network system could use virtual switch to chain network functions on the traditional middleboxes or on the virtual machines (VMs) in the commodity [1] servers for a SFC as shown in Figure 1b. The use of VNFs improves the system performance compared to the NOS solution and increases the system flexibility [3], [4] compared to the traditional solution with fixed network function implementations. However, SDN switches is designed to support flow-based forwarding by examining packet header fields [5] and it can not differentiate the packets to steer network traffics for a SFC without packet header modifications [6].

This paper proposes a VNFs-enabled switch, as shown in Figure 1c, to address the additional overhead when using SDN switch to steer network traffics for different SFCs and to maintain the system flexibility when leveraging the many-field matching results. The network functions are embedded as switch's extension modules and could be enabled in an arbitrary order for effective packet processing and flexible traffic steering. The proposed solution eliminates the cost of header modification by using the matching results to call the software modules in specific order for a SFC. The proposed solution also reduces the number of matching for each incoming packet by combining required matching process from different NFs. Moreover, the proposed VNF-enabled switch

avoids the unnecessary data movement between different NFs to improve the system performance by setting common buffers for different NFs. In summary, this paper has three main contributions: 1) scalability: the proposed system removes the unnecessary matching process for packet forwarding actions between different NFs without the need of data movement to increase the system scalability; 2) flexibility: when a new service chain is created, the system just simply changes the flow table to enable different network function modules; 3) integration: it integrates the concepts of SDN and NFVs and remove the additional overhead, which is an alternative way to implement network functions that examine both header fields and payload.

The remainder of this paper is organized as follows. The related works are discussed at Section II to demonstrate the trend to leverage the available headroom in network devices. The forwarding challenges to use SDN and NFV to construct service chains are discussed at Section III. The proposed VNF-enabled SDN switch and its components are discussed at Section IV to address these challenges. Section V demonstrates the experimental results to show the effectiveness of the proposed system. Finally, this paper is concluded at Section VI.

II. Background and Related Works

Due to the decoupling of control and data planes in the SDN network devices, more computation resources in the network devices are now spared and could be used for new network features. In this section, two major design directions of the SDN network devices are discussed to demonstrate the possibilities to implement new features in network devices to construct a more efficient network.

A. Reconfigurable Switch

OpenFlow protocol is the current standard of SDN network. Over the past few years, the specification of OpenFlow protocol becomes more complex and incorporates more header fields to support more applications. In order to mitigate the challenge caused by the increasing number of fields, several studies have been proposed [7], [8], [9]. These works focus on how to configure the data plane to make sure new headers in the packets of a flow will be parsed correctly based on the commands giving by the controllers. These designs increase the flexibility of the network by adding new features into SDN network devices.

In addition, several research works focus on the network function implementations for a better system flexibility. For example, [10] design a software-defined device framework for network functions development, deployment, and management. [11] designs a NFV platform to support large numbers of lightweights NFs. Both of them divide a network function into a set of processes and propose the best way to implement these processes to work as a whole network function.

B. Consolidated Middlebox

Middleboxes are hardware devices attached to a network to support the needs to have additional functions to the network. The network function running on traditional middleboxes are normally fixed and can not be upgraded easily. Moreover, these middleboxes are placed and connected to a network physically and the re-configuration of those middleboxes is costly. To address the configuration and functionality problems, NFV follows the similar system architecture but runs on the virtual platforms. These VNFs could be initiated in the needed-base to lower the capital expense and operation cost. Both network function implementation approaches allow the third parties to develop their own solutions and insert in the network when it is needed. Several studies have been proposed [12], [13], [14] to leverage the headroom in network device and run applications directly on network devices. These studies focus on how to integrate processing components into the network for a better resource sharing and a higher system utilization.

Moreover, the advance of server technology makes it possible to run a scalable software switch on a single high volume commodity server or across multiple servers with 4-6 NICs, where each NIC fits upto 8 ports [15]. The advance of virtualization technology on commodity server also makes it attractive to put VNFs on a server [16][17]. Hence, it becomes economic to integrate both SDN switches and VNFs within a single commodity server.

III. SFC Packet Forwarding

SDN is powerful to direct network traffics for different service chains, but it comes at a cost for application-based forwarding due to the necessity of header modification [6]. In this section, the challenges when using SDN to direct network traffics to several networks in service chains are discussed and demonstrated. To address these challenges, the proposed solution is discussed in the next section.

A. Header Modification for Packet Forwarding

In order to support the application-based forwarding for a service chain, SDN switches require the modifications on each packet header to track the packet's processing status in a service chain. Hence, it requires the SDN switches to conducts the matching process on all incoming packets. This requirement results multiple matchings on the same packet. Both packet header modification for traffic steering and multiple matching on the same packet are expensive and affect the system performance as shown in Figure 2.

B. Multiple Matchings for Packet Forwarding in a SFC

The network switch conducts simple and straight forwarding actions. When a packet arrives at a switch, the switch initiates the matching process to determine where a packet should be forward to without knowing whether a packet's status. To forward the same packet to the different network functions in a service chain, several header fields are kept the same for matching process to make sure the identification of a packet. Only a few header fields need to be updated accordingly for the purpose of traffic steering. Hence, the same packet will be verified multiple times. As shown in Figure 3, those fields used to distinguish packets need to be verified $n + 1$ times at the

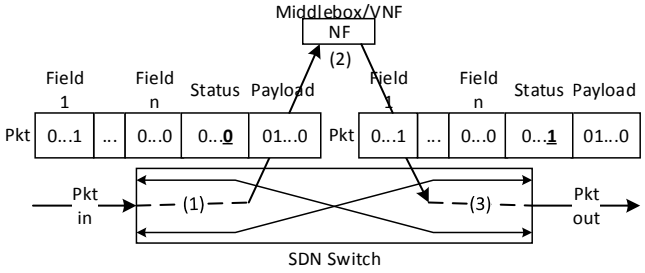


Fig. 2: An example to demonstrate the traffic steering costs when using SDN to direct network traffics to a NF. A SDN conducts the matching process twice against the same packet in step (1), (3) and requires the header modification to update the processing status in step (2).

switch for a SFC with n NFs. Moreover, there are n updates in the header field of a packet for traffic steering. Both of them are wastages and can be avoided if NFs could be conducted in a specific order based on the first matching results.

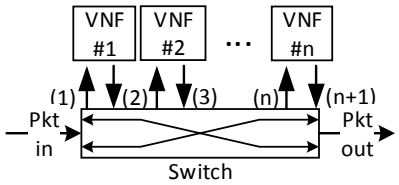


Fig. 3: An example to demonstrate how a packet is forwarded between different NFs with a SDN switch. If a switch is connected with n VNFs, the switch need to conduct matching process $n + 1$ times on the same packet.

C. Redundant Data for NFV Cooperation

The independent of each network function makes the sharing of data between different network functions difficult. To guarantee the correctness of distributed operations, the shared data to different middleboxes needs to be synced for all boxes [18]. As shown in Figure 4, two VNFs are implemented on different virtual machines in a commodity server. Since two VMs are isolated, each VNF needs to have its own copy of store packet data in the memory. However, it is costly to generate additional copies of data and distribute them into different NFs. To eliminate the redundant data, it is necessary to share and reuse packet data among NFs.

IV. NF-Switch

In the previous section, the forwarding challenges using SDN to steer network traffics to pass through different NFs are discussed to show the possible bottleneck of the current NF implementations with SDN. In this section, the proposed VNF-enabled SDN switch and its key components are discussed.

A. Switch Architecture

A VNF-enabled SDN switch architecture is proposed as Figure 5 to reduce the unnecessary packet matching as discussed

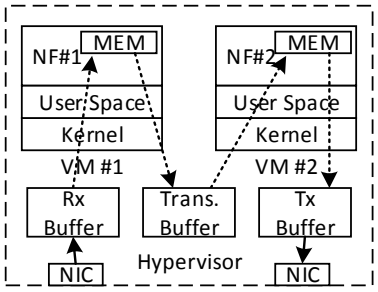


Fig. 4: An example to demonstrate how two independent VMs can cooperate with each other by duplicating data for each node with buffers in the hypervisor.

in Section III and to improve the sharing of data between VNFs. The receiving module Rx gets packet from previous node for the following packet processing. When the packet processing is done, the transmitting module Tx forwards it to the next node. All VNFs are designed as software modules to the switch and are called in the needed-base. Those NF modules could be called in any sequence since they are just software module. Moreover, the data buffer and the processing states are stored in the shared memory and are accessible to all modules. The NF modules are conducted in order and there is no data contention between different modules.

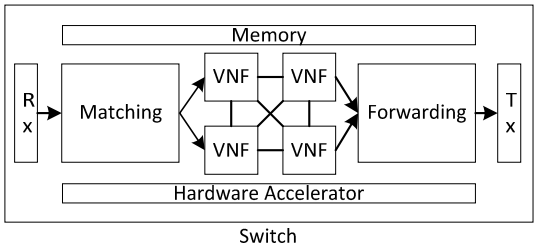


Fig. 5: The proposed VNF-enabled SDN switch with VNFs as software modules that can be called in an arbitrary order.

In order to reduce the redundant copies of a packet between different network functions, the packet buffer and the state buffer are designed in the proposed system as shown in Figure 6. Both of them are shared among all modules and can be accessed by different network functions. The packet buffer keeps the packet data and state buffer keeps the processing state of each packet. All VNFs module are conducted in a specific order based the required service and different flows are with different buffer locations. Hence, there is no contention between different modules. Once a packet passed though all network functions, the data in the state buffer is attached to the pre-defined field in a packet in order to share the processing status between different switches.

B. Network Function Consolidation

To implement network functions onto a switch, each network function has to be coded as a module and pre-installed into a switch. Moreover, it becomes cost effectively to have

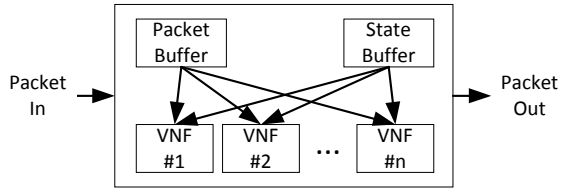


Fig. 6: The implementation of shared memory in VNF-enabled SDN switch.

one core process for each network function and use the shared supportive processes between all network functions in a switch. The consolidation of multiple network functions provides the additional design space to combine some common processes from different network functions to remove the common waste.

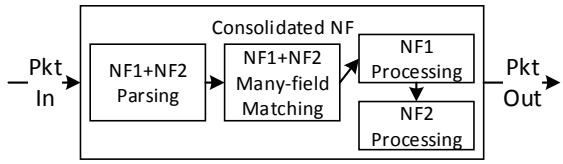


Fig. 7: An example to a service chains using the consolidated network functions to reduce the costs on parsing and matching.

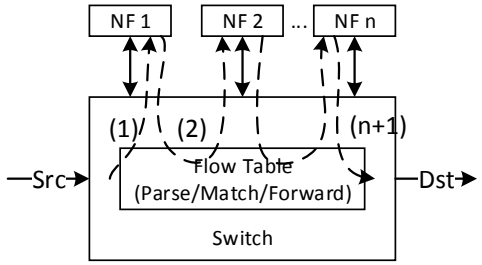
Instead of fetching packet header fields for different matching processes, it is cost effectively to fetch all targeted header fields at the same time and conduct the matchings against all those field to avoid I/O overhead and additional matching cost. SDN switch is specified to support many-field packet classification and more and more field are incorporated into SDN switch's specification. By leveraging this new feature, it become possible to define a forwarding rule that contains all required field for both NF1 and NF2 for supportive processes like parsing and matching. The core processes of NF1 and NF2 are then initiated in the required sequence for a service chain based on the matching result as shown in Figure 7.

V. Experimental Results

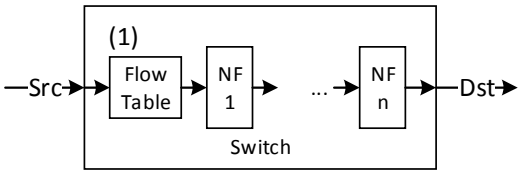
In this section, the effectiveness of the proposed solution is verified on both single server and on a simulated network. The SFC content switching time is first verified to show the flexibility of the proposed system. The forwarding latency reduction and system throughput improvement in a network are then discussed to demonstrate how the proposed system can work with different service chains.

A. Experiment Setup

All experiments are conducted on a Linux machine running Ubuntu 16.04 with two Intel Xeon E5410 CPUs (2 x 4 cores) and 48GB memory, QEMU 2.5.0 and OVS 2.7.0. To eliminate the possible delay cause by the external network links between switch and VNFs, both virtual switches and VNFs run on the same server for the experiments. However, the proposed system is not limited to a single server. Both switches and



(a) The SDN+NFV setting



(b) The proposed VNF-embedded OVS

Fig. 8: The demonstration of three experiment designs. In (a), the switch has to conduct n+1 times of matchings in order to forward a packet for service chain. In Figure (b), the proposed solution leverages the many-field flow table and required only one matching to construct the same service chains.

VNFs could be located in different geographic locations. To simulate the network at the large scale, the Mininet [19] is used to evaluate the proposed solutions. To measure the system performance of the proposed switch design, iperf [20] applications are implemented. Moreover, the worst case scenario is assumed where all packets need to be forwarded to the required network functions in a service chain. The high level of the experiment design is shown as Figure 8.

B. System Flexibility

By embedding network functions into a switch, the network system becomes flexible to construct different SFCs using the pre-defined network functions. Instead of modifying multiple forwarding rules to steer network traffics for a new SFC as in the traditional solution, the proposed solution uses the matching result to initiate the network function to avoid multiple matching and I/O costs. To show system flexibility to switch between different SFCs, *SFC content switching time* is defined to measure the required time to construct a new SFC for an existing flow in a switch. The SFC content switching time measures the additional elapsed time of the first packet in the same flow after the implemented SFC is updated. To make it a fair comparison, it is assumed that the two SFCs running the same network functions but in different order.

As shown in Figure 9, the SFC content switching time is proportional to the number of NFs in a SFC for the traditional setting requires multiple forwarding table updates and packet matchings. In the traditional solution, the matching to steer traffic for different network functions in a switch and the Middlebox/VM I/O costs result in a longer SFC content switching time. However, the SFC content switching time is almost the same for the proposed solution due to only

one forwarding table update and one matching is required for SFCs with different number of network functions. The proposed solution reduces the SFC content switching time by embedding NF in to switch to remove I/O costs and leverage the many-field packet classification result to reduce the number of matchings for traffic steering.

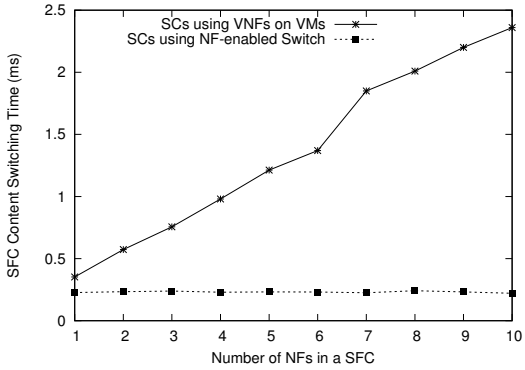


Fig. 9: The comparison of SFC content switching time between the traditional solution and the proposed NF-enabled switch. The traditional solution steers network traffics with the help of SDN switch and it is required for multiple matchings after an implemented SFC is updated. The proposed solution leverages the many-field flow table and calls the pre-defined network functions to construct a SFC.

C. System Performance on a Single Server

In this section, the proposed solution is discussed and compared with the traditional solution using VMs to show the improvement of the proposed solution. Both the system bandwidth and the processing latency are discussed to show the impact of the usage of SDN switch to construct service chains with multiple network function on a server and how the proposed solution improves the system performance.

1) Bandwidth Comparison: When using a SDN switch to connect different network functions running on VMs for a SFC, the same packet has to visit the SDN switch multiple times. When more network function running on VMs are attached the same switch, a switch has to spend more bandwidth on the same packet and results in low system scalability. As shown in Figure 10, the SFC throughput is decreasing when there are more network functions in a SFC. The I/O costs of different VMs and the matching costs to steer network traffic limit the available bandwidth of a system.

By embedding network functions into a switch, the switch needs one matching for each packet, which save the I/O cost and the duplicated matchings on the same packet and improve the system bandwidth. To increase the bandwidth for a SFC, the load-balancer solution has a slightly different network topology by adding distributor and collector to increase the available bandwidth for a SFC by shifting the computation to additional NF-enabled switches. The load-balancing solution results in a lower SFC throughput when the number of NFs in a SFC is small due to additional load-balancing component.

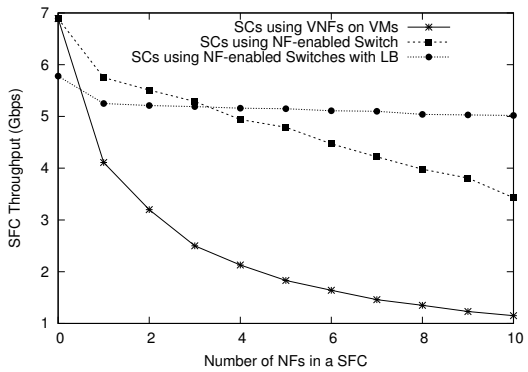


Fig. 10: The bandwidth comparison between the traditional service chains using VMs and the proposed NF-enabled switch.

Due to additional NF-enabled switches running in parallel, it has a higher SFC throughput with the increasing number of NFs.

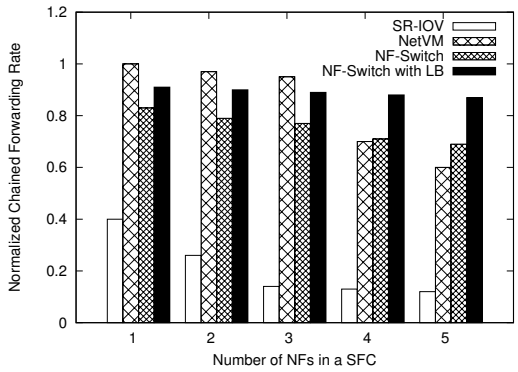


Fig. 11: The normalized chained forwarding rate between NetVM [21], SR-IOV, NF-Switch, and NF-Switch with load balancer.

Moreover, NF-Switch and its load-balanced implementations are compared with NetVM [21] and SR-IOV implementations in term of the normalized chained forwarding rate as shown in Figure 11. To connect different NFs for a SFC, a software switch is used in a server. When there are only a few NFs in a SFC, the NetVM performs better due to the shared memory for the zero data copy between NFs on VMs and a lower network traffics on the switch. However, it generates more network traffics of a SFC when there are more NFs in it. The excessive network traffics of a SFC on the switch results in a lower bandwidth for a SFC. The proposed NF-Switch is able to maintain the chained forwarding rate even with more NFs in a SFC by removing the I/O cost and the duplicated data between NFs in a SFC.

2) Latency Comparison: As shown in Figure 12, the processing latency increases dramatically with a longer service chain for the traditional solution when running NFs on VMs. To steer network traffics to visit different NFs on VMs for a service chain, the processing latency is increased due to the I/O

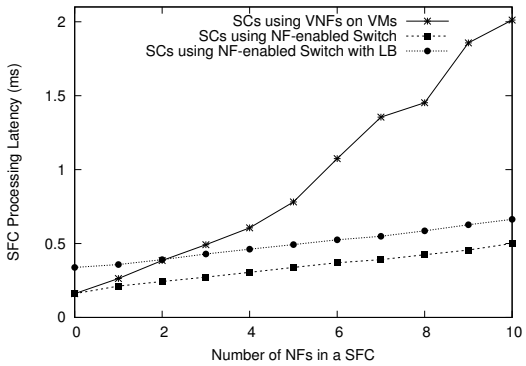


Fig. 12: The latency comparison between the service chains using VNFs on VMs and the proposed NF-enabled switch.

Solutions	The traditional NFV implementation with VMs				The proposed solution	
# of NFs	3 NFs		6 NFs		3 NFs	6 NFs
NF Distribution	Single	Distr.	Single	Distr.	Distr.	Distr.
SFC Content Switching(ms)	1.80	1.69	1.14	0.94	0.39	0.41
Latency (ms)	0.69	0.50	1.12	0.71	0.16	0.26
Bandwidth (Gbps)	3.04	3.14	1.92	1.98	11.4	10.3

TABLE I: The latency and bandwidth comparisons between the traditional service chain implementation with virtual machines in a network and the proposed NF-enabled switch.

transmission delay of each VM. Moreover, each packet also requires multiple packet matching on the switch in order to reach different NFs in the required order. With a longer service chain, more packets are waited in the queue and this makes the processing latency even longer compared to a shorter service chain. By embedding network functions into a switch, the I/O cost to visit different VMs is removed and the network traffics in a switch becomes less since each packet does not need to visit the switch multiple times for traffic steering between NFs.

D. System Performance on a network

The proposed NF-enabled switch could also be used to improve the network performance by reducing the I/O cost and distribute NF to different switches on the same route in a network under the coordination from NOS. As expected, the distributed traditional implementations have a better performance due the less resource competition at the same node. Compared to the traditional solution, the proposed NF-enabled not only improve the processing latency but also maintain the system bandwidth. Moreover, the proposed solution can save the required content switching time when applying a SFC to a flow as shown in Table I for a better system flexibility.

VI. Conclusion

Working together, SDN and NFV create a network with a high manageability and a lower operation/capital cost for the system administration. However, the traditional way to attach network functions to a existing network generates new challenges such as the duplicated operations between different network functions, the redundant packet data distribution in packet processing, and the excessive traffic in a network with co-existing service chains. In this paper, a NFs-enabled switch is proposed to address these challenges by combining duplicated matching processes from different network functions together and by reducing the costly packet header modifications.

The heterogeneity of active modules in each switch could be a good target of hardware accelerator. Currently, this part is not addressed in this paper. Our future work will focus on the improvement of system performance by leveraging hardware accelerators such as GPU, FPGA, or multi-core platform. Another direction of our future work will focus on the varying NF modules and the standard interface to load/unload different NF modules.

References

[1] J. Martins and et al, "Clickos and the art of network function virtualization," in *Proc. USENIX NSDI*, 2014, pp. 459–473.
[2] P. Berde and et al, "Onos: Towards an open, distributed sdn os," in *Proc. ACM HOTSDN*, 2014, pp. 1–6.
[3] *Case Study: Industry Largest NFV Deployment*, Bigswitch Networks, 2015, http://go.bigswitch.com/rs/974-WXR-561/.
[4] "Noviflow," https://noviflow.com/switches-for-nfv/.
[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM CCR*, vol. 38, no. 2, pp. 69–74, 2008.
[6] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," in *Proc. ACM SIGCOMM*, 2013, pp. 27–38.
[7] P. Bosshart and et al, "P4: Programming protocol-independent packet processors," *SIGCOMM CCR*, vol. 44, pp. 87–95, 2014.
[8] L. Jose, L. Yan, G. Varghese, and N. McKeown, "Compiling packet programs to reconfigurable switches," in *Proc. USENIX NSDI*, 2015, pp. 103–115.
[9] H. Song, "Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane," in *Proc. ACM HOTSDN*, 2013, pp. 127–132.
[10] A. Bremler-Barr, Y. Harchol, and D. Hay, "Openbox: A software-defined framework for developing, deploying, and managing network functions," in *Proc. ACM SIGCOMM*, 2016, pp. 511–524.
[11] W. Zhang, J. Hwang, S. Rajagopalan, K. Ramakrishnan, and T. Wood, "Flurries: Countless fine-grained nfs for flexible per-flow customization," in *Proc. CoNEXT*, 2016, pp. 3–17.
[12] V. Sekar, N. Egi, S. Ratnasamy, M. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proc. USENIX NSDI*, 2012, pp. 24–37.
[13] J. Lee, J. Tourrilhes, P. Sharma, and S. Banerjee, "No more middlebox: Integrate processing into network," in *Proc. ACM SIGCOMM*, 2010, pp. 459–460.
[14] T. Park, Y. Kim, and S. Shin, "Unisafe: A union of security actions for software switches," in *Proc. ACM SiSDNNFV*, 2016, pp. 13–18.
[15] M. Dobrescu and et al, "Routebricks: Exploiting parallelism to scale software routers," in *Proc. ACM SOSP*, 2009, pp. 15–28.
[16] A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Koral, "Deep packet inspection as a service," in *Proc. ACM ICENET*, 2014, pp. 271–282.
[17] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker, "E2: A framework for nfv applications," in *Proc. SOSP*, 2015, pp. 121–136.
[18] Gember-Jacobson and et al, "Opennf: Enabling innovation in network function control," in *Proc. ACM SIGCOMM*, 2014, pp. 163–174.
[19] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. ACM Hot Topics*, 2010, pp. 19:1–19:6.
[20] "Iperf," https://iperf.fr/.
[21] J. Hwang, K. K. Ramakrishnan, and T. Wood, "Netvm: High performance and flexible networking using virtualization on commodity platforms," in *Proc. NSDI*, 2014, pp. 445–458.