Separating VNF and Network Control for Hardware-Acceleration of SDN/NFV Architecture

Tong Duan, Julong Lan, Yuxiang Hu, and Penghao Sun

A hardware-acceleration architecture that separates virtual network functions (VNFs) and network control (called HSN) is proposed to solve the mismatch between the simple flow steering requirements and strong packet processing abilities of software-defined networking (SDN) forwarding elements (FEs) in SDN/network function virtualization (NFV) architecture, while improving the efficiency of NFV infrastructure and the performance of network-intensive functions. HSN makes full use of FEs and accelerates VNFs through two mechanisms: (1) separation of traffic steering and packet processing in the FEs; (2) separation of SDN and NFV control in the FEs. Our HSN prototype, built on NetFPGA-10G, demonstrates that the processing performance can be greatly improved with only a small modification of the traditional SDN/NFV architecture.

Keywords: NFV, SDN, Hardware-acceleration, Network function chaining, NetFPGA.

I. Introduction

Network function virtualization (NFV) enables traditional packet processing to move from dedicated hardware-based devices to virtual machines (VMs) located on pools of commodity servers, which introduces great flexibility and portability to network function (NF) designs. Because of such advantages, NFV is regarded as a natural complementary technology of software-defined networking (SDN), in which the network programmability of SDN offers the ability for network connectivity, while the flexible instantiation of virtual NFs (VNFs) provides the equivalent capability for functional processing. As shown in Fig. 1(a), this SDN/NFV architecture, standardized by open networking foundation [1], leverages the complementary interaction between SDN and NFV, thus it has attracted enormous attention from both researchers and manufacturers. It has also been extended to many research areas such as VNF deployment algorithm design and orchestration system design. However, it has two main disadvantages:

(1) Waste of the strong processing ability of SDN forwarding elements (FEs). In SDN/NFV, the contribution of SDN remains in the network infrastructure domain of the NFV infrastructure (NFVI), focusing on providing flexible connectivity services. However, the commonly-used SDN data plane abstraction – OpenFlow, a processing pipeline with 12-tuple fields or more for matching and several programmable instructions for action, has a considerably more processing ability than just forwarding, let alone the more flexible data plane abstractions like protocol-oblivious forwarding (POF) [2] and protocol-independent packet processors (P4) [3]. Many kinds of stateless L2/L3 processing like Firewall and network address translation (NAT) can be easily executed by SDN FEs, but this strong ability may be wasted in the traditional SDN/NFV architecture.

Manuscript received Mar. 9, 2017; revised May 15, 2017; accepted June 7, 2017. This work was supported by the National Natural Science Foundation of China (973), China (Grant No. 61521003, 61372121), and the National High Technology Research and Development Program of China (863), China (Grant No. 2015AA016102, 2013AA013505).

Tong Duan (corresponding author, duantong21@outlook.com), Julong Lan (ndsclj@163.com), Yuxiang Hu (chxachxa@126.com), and Penghao Sun (sphshine@126.com) are with the Department of Communications, National Digital Switching System and Technology Research Center (NDSC), Zhengzhou, China.

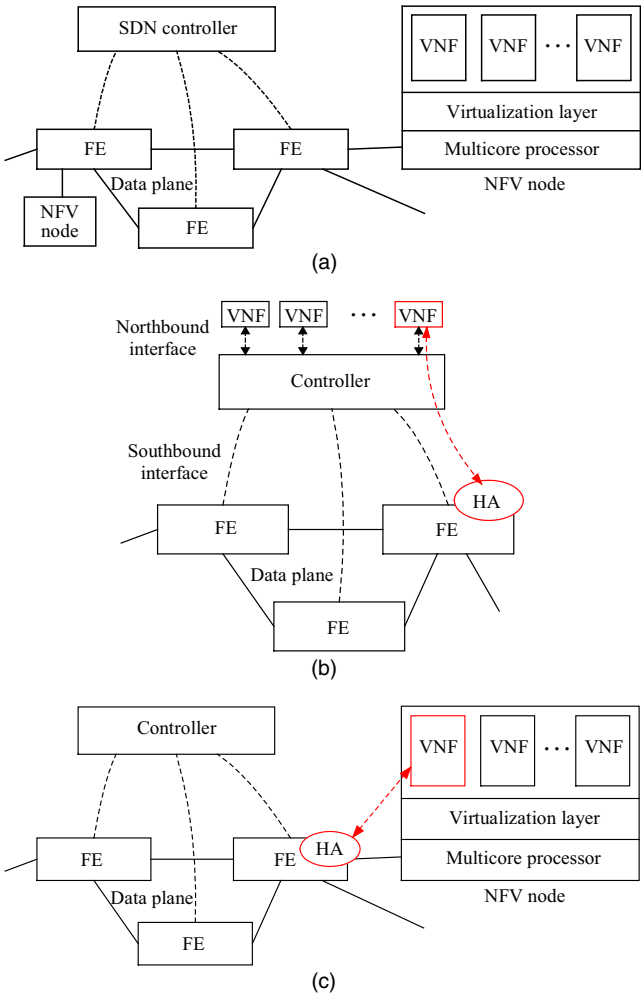

Fig. 1. SDN/NFV architecture and HA mechanisms: (a) traditional SDN/NFV Architecture; (b) controller-based NFV HA; (c) VM-based NFV HA.

(2) Low performance levels of software-based VNFs. A typical NFV server is composed of a processor platform, a virtualization layer, and several VNF instances. This node exposes several performance bottlenecks compared to hardware-based NFs [4]: a) the overhead of the virtualization layer that is built on a hypervisor and virtual switch limits the packet transmission rate between the processor platform and VM; b) the VM operating system overhead and kernel networking stack limit the packet processing performance; c) CPU processing abilities limit the processing rate (such as DRAM accessing) of network-intensive (NI) functions while handling a large number of packets. Thus, hardware acceleration (HA) is a technology that would enable NFV to guarantee high performance.

To solve these two main disadvantages of SDN/NFV, a natural idea is what we call FE-enabled HA: offload stateless VNF processing into FEs, making full use of the FE processing resources while accelerating the NI functions. This FE-enabled HA mechanism has several drawbacks: on the one hand, combining steering and VNF processing in one FE will confuse the control plane task of the SDN controller and VNF manager; on the other hand, FEs must play a dual role for both traffic steering and VNF processing, which easily leads to processing rule conflicts in the same flow table. In this paper, however, we argue that the problems of SDN/NFV architecture mentioned above are mainly caused by the mismatch between the simple flow steering requirements and the strong packet processing abilities of SDN FEs. Based on this observation, we propose a Hardware-acceleration architecture that Separates VNF and Network control (HSN) for SDN/NFV. HSN makes full use of FEs and accelerates VNFs through two mechanisms:

(1) Separated traffic steering and packet processing. In traditional FE-enabled HA architecture, traffic steering, and packet processing are mixed in one FE; but in HSN, these two parts are clearly divided so that not only can an FE's processing ability be fully used but rule conflicts can also be avoided.

(2) Independent control of SDN and NFV. In HSN, every FE exposes itself to the SDN controller and HA manager (HAM) via different interfaces so that VNFs only need to deal with the packet processing part and SDN only provides the network connectivity of the NFVI.

By clearly dividing the mission of the SDN and NFV, HSN can achieve better HA performance than existing methods. We built a prototype on NetFPGA-10G platform with two SDN controllers, one of which plays the role of an FE controller and the other one acts as a HAM. Evaluation results show that the forwarding performance is increased by 30% while only a few extra resources are used.

In the next section, we introduce the related work; then we describe the HSN architecture and detail the HSN FE structure, including both hardware and software design. In the following section, we evaluate the performance of a HSN prototype platform using the Spirent TestCenter by injecting into it IPv4 traffic. Finally, we conclude the paper.

II. Related Work

To solve the two main disadvantages of SDN/NFV mentioned in Section I, a natural idea is to offload stateless VNF processing into FEs, making full use of the FE processing ability while accelerating NI functions. References [5], [6] even offload some stateful processing (such as TCP connectivity) to FEs. One step further, the deployment and routing mechanisms based

https://doi.org/10.4218/etrij.17.0117.0174

on this idea have also been studied in [6], [7]. However, whether the offloaded VNF logic in FEs should be controlled by the FE controller or VNF still needs further discussion. One method used in [4], [5] is to build the VNF on an FE controller using a northbound interface, as shown in Fig. 1(b). In this architecture, each VNF is logically divided into two parts: one part is located in the FEs and the other is the associated function app running on the SDN controller. The drawback of this method is obvious: most of the VNFs are running on top of the controller together with several other network managing apps, which gives both network operator and VNF manager roles to the controller. As a result, the SDN controller can be easily overloaded.

The other method suggests that the offloaded VNF logic in FEs be configured by VNFs built on common servers [8], as shown in Fig. 1(c). In this method, the partial VNF processing logic in FEs is independent of the SDN forwarding logic. However, in this method, the forwarding rules and VNF rules are mixed together in the FE's match table, which will easily lead to rule conflicts. Moreover, because the VNF rules are configured by VNFs, it is extremely hard for an SDN controller to handle these rule conflicts.

Another idea is to directly accelerate VNFs in NFV nodes, without offloading processing to FEs. This idea is always realized by using accelerating cards such as field-programmable gate arrays (FPGAs) [9], [10], which can flexibly implement customized processing pipelines for different VNFs. Further, in this situation, as suggested by [11], [12], the SDN and NFV control can be separated by using different controller, enabling the flexibility of mapping HA rules into NFV nodes. However, this idea does not change the traditional SDN/NFV architecture but only focuses on NFV node acceleration, thus it is not within the scope of this paper.

III. HSN Architecture

In this section, we propose the HSN architecture and then detail the FE structure design and HA control interface.

1. Overall Architecture

The overall HSN architecture is shown in Fig. 2. As we can see, HSN is nearly the same as what is set out in the NFV architectural framework [13], but there are two main differences: (1) each FE is divided into two main parts – the forwarding (FW) part and HA part – and each part is configured by different managers; (2) HAM(s) is/are added into the virtual infrastructure manager (VIM) to control the HA logic in the FEs. In traditional SDN/NFV architecture, VIM is in charge of compute and hypervisor control (CHC) and network control (the FE controller, typically an SDN controller), allocating corresponding resources to VNFs that are instantiated and managed by the VNF manager (VNFM). VIM also interacts with the NFV orchestrator (NFVO) to provision NF chains (NFC) for network services. However, in HSN, VIM is also responsible for allocating proper HA resources for VNFs that have specific acceleration requirements, and in this situation, a HAM should configure the HA part of the FEs for VNF offloading, while an FE controller only ensures the network connectivity and traffic steering between VNFs.

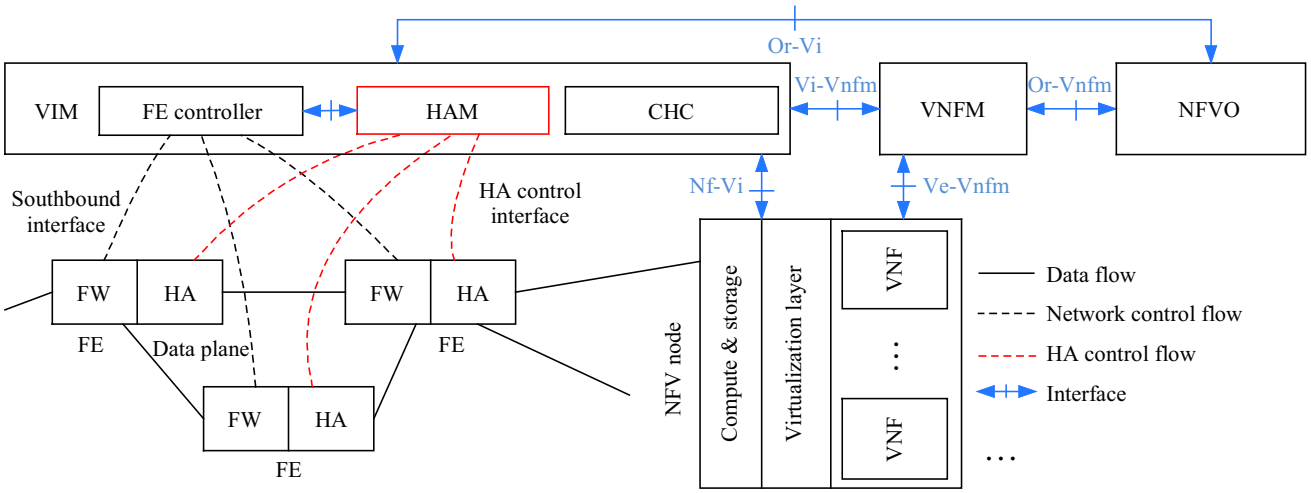

Fig. 2. HSN architecture.

In HSN, the Vi-Vnfm interface is also extended to allow acceleration requirements to be described in the VNF description (VNFD), which includes the deployment configuration and operational behavior of a VNF and post-deployment capabilities. Then, HAM allocates and binds an FE HA to a VNF based on HA resource availability and capability and the VNF requirements, which include acceleration type and number of operation rules. HA resources of the same FE may be shared among several VNFs. After the HA allocation by the HAM, the VNF configuration information (such as match rules and action instructions) will be exchanged between the VNF and HA to define the specific acceleration operation. The HAM also exchanges the VNF–HA mapping information with the FE controller; thus, the FE controller can use this information for traffic steering optimization.

2. FE Structure

Typically, the FE structure is built based on OpenFlow abstraction, but if more programmability is required, protocol-independent structures such as POF or P4 [14] can also be used. As shown in Fig. 3, in HSN, each FE has nearly the same structure as an OpenFlow switch except for a few changes.

The first change is that a classifier component is added to ensure network-wide policy enforcement. Generally, to ensure that a packet flow is processed by the right order of NFs, some tags need to be carried in packet headers to indicate whether the packet has been processed by the right VNF [15], [16]. These tags will decide the behavior of a classifier. If a packet flow should be processed by a VNF, and this VNF is just accelerated in the FE, then the classifier will first direct the traffic to be operated by the HA table and then be forwarded to the right port, as shown by arrow ① of Fig. 3. If the FE does not support this VNF acceleration, then the classifier will direct the traffic to the FW table, as shown by arrow ② of Fig. 3.

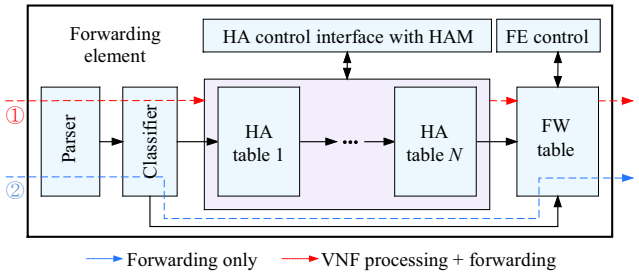

Fig. 3. FE structure.

Another change is the separation between the HA and FW tables. The FE match tables (each typically consists of a match table and an action processor) are divided into a forwarding (FW) part and an HA part, and each part is configured by different managers. This independent control is designed to avoid rule conflicts or controller overload caused by existing mechanisms. Based on the observation that the processing demands vary between different VNFs but the forwarding demands of traffic steering are usually fixed, the HA table is designed with more flexibility and occupies more resources than the FW table. In HSN, the match fields of each HA table can be flexibly selected by the match selector and the action instructions can be configured at bit-granularity to meet a variety of VNF requirements, while the FW table only matches limited fields (such as 5-tuple headers) and performs simple forwarding instructions. Through proper configuration, two or more HA tables can be logically composed for one VNF processing; and through a go-to instruction, two or more VNF processing can be operated sequentially between logical HA tables in the same FE.

3. HA Control Interface

The separation between the HA and FW tables needs to be supported by separated controls. In HSN, the FE controller only handles the network-related state, ensuring network connectivity and traffic steering, while the HAM only handles VNF-related messages. In the data plane, this separation can be executed by the HA and FW tables: only the FW table can direct packets to the FE controller for forwarding path installation and HA tables only perform the match and action operation and will never send the packet to the controller or HAM, even if no entries match. In the control plane, the HAM does not handle any packet-in messages. It only performs the HA allocation and configuration. This separation is beneficial to network-wide policy enforcement, while significantly reducing the FE controller load.

The HA control interface only has two types of messages:

- Status Enquiry. HAM uses this message to enquire about the total and idle resources of the HA tables, which is necessary for optimal mapping between VNF and HA tables.
- HA Configuration. The HAM uses this type of message to configure the VNF processing logic into HA tables under the instruction of VNFD. It consists of configuring the match selector and writing or deleting the match rules and action instructions of each HA table.

https://doi.org/10.4218/etrij.17.0117.0174

IV. NF Chaining in HSN

A network service is usually composed of one or more NFs, which can be realized by routing data traffic through a series of VNFs. This service provisioning scheme is called NF chaining (NFC), and is one of the original design purposes of SDN/NFV architecture. It mainly consists of two steps: VNF deployment and traffic steering. The VNF deployment process instantiates VNFs in optimal locations to optimize resource and bandwidth occupancy. In this section, we do not discuss this process since several algorithms have been proposed to address this problem, and we assume the VNFs have been deployed in the optimal locations. We only deal with how to allocate HA resources to VNFs with specific acceleration requirements and how to route traffic through these VNFs.

1. VNF–HA Mapping and Traffic Routing

Considering there is always a large amount of continuous traffic to be processed in an NI VNF, the mapping between the HA and VNF is static with no change in the HA allocation during the VNF life cycle. In HSN, the processing in the FE is stateless, thus while mapping a VNF that requires a stateful connection, the HA can only be allocated in the FE that connects with the NFV node where the VNF is located. However, for a stateless VNF, HA can be instantiated at any location as long as it meets the resource requirements. VNF–HA mapping using stateful FE will be left for further discussion.

After the VNFs are deployed in the right NFV nodes and HA is instantiated in the right FEs, the subsequent traffic steering should take VNF–HA mapping into consideration. As HA located in the FEs can offload a large amount of traffic passing through NI VNFs, the design principle of path routing is to process the traffic in the FEs preferentially. For example, as shown in Fig. 4, when routing a packet flow through a NFC of {VNF A, VNF C, VNF E} from a user to a server, considering that these three VNFs all have a HA logic in the FEs, the preferable path should be FE3 → FE1 → FE2. Without HA, the routing length and complexity will rise rapidly: FE3 → FE1 → NFV Node1 → FE1 → FE2 → NFV Node2 → FE2.

2. HA Table Mapping

Once a specific VNF is determined to be mapped into a specific FE, the hardware HA table should be carefully configured to perform the corresponding VNF operation. Each VNF has a specific requirement including the space of rules, type of match fields, and action instructions. The

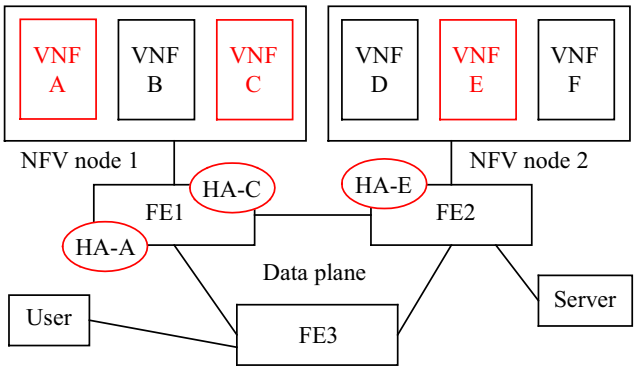

Fig. 4. HSN NFC use case.

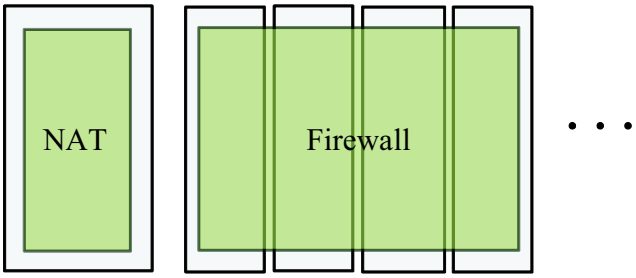

Fig. 5. Example of HA table allocation.

detailed HA table mapping methods are inspired by [17], and there are mainly two situations: a) one table for one VNF and b) several tables for one VNF. For example, a NAT VNF may need 200 HA rules to match a 32-bit IP_src and to perform the modify_IPsrc instruction. Assuming that the match width of each HA table is exactly 32 bits and its depth is more than 200, then one HA table is enough for this VNF, as shown in Fig. 5. If the match width of a VNF (the 5-tuple Firewall in Fig. 5) is larger than one HA table, then several HA tables should be combined for VNF matching. Another situation occurs when the required match depth of one VNF is larger than the depth of one HA table; then, it would require several HA tables to store and match the VNF rules. This can be achieved by a proper configuration of the metadata transmitting through different HA tables.

In HSN, because of the large cost of HA table resources and limited hardware resources, a practical problem of HA tables is how to select the appropriate match width and depth of each HA table, and this problem should be carefully evaluated. To formally describe this problem, we assume that there are typically K VNFs to be mapped and the required match depth and width of each VNF is denoted as d_i and w_i, $1 \le i \le K$, respectively. We aim to use as few HA table resources as possible. The problem can be formulated as follows:

$$\text{min: } C_{\mathrm{HA}}(W,D) = N_{\mathrm{HA}} \times \Big(C_{\mathrm{S}}(W) + C_{\mathrm{M}}(W,D) + C_{\mathrm{A}}\Big), \quad (1)$$

$$\text{subject to: } N_{\mathrm{HA}} \geq \sum_{i=1}^{K} \langle w_i/W \rangle \times \langle d_i/D \rangle, \quad (2)$$

where

- C_{HA}: total HA table cost, including the cost of the match selector, match table, and action processor
- N_{HA}: total number of HA tables in one FE node
- W: the width of each match table
- D: the depth of each match table
- C_{S}: cost of match selector, depending on match width W and header vector length L (L is constant)
- C_{M}: cost of match tables, depending on match width W and depth D
- C_{A}: cost of the action processor of each HA table

Inequality (2) indicates that the number of HA tables should be larger than the required HA resources, in which function $\langle x/y \rangle$ denotes rounding up the value of x/y.

Generally, function $C_{\mathrm{M}}(W, D)$ is not linear with respect to the two parameters, thus for different VNFs set and different implementation platforms, the optimal W and D would be different. Generally, the hardware implementation should consider the most representative and commonly used VNFs, and carefully select suitable HA table volumes.

3. Mathematical Model and Problem Definition

Based on the details of VNF-HA mapping mentioned above, we introduce a mathematical model for our system and formally define the VNF-HA mapping and traffic routing problem.

Network Topology. The physical network is represented as an undirected graph $G = (N, L)$, where N and L denote the set of FE nodes and links, respectively. As defined in the HSN FE structure, each FE node $n \in N$ has HA tables to hold the VNF rules, and the idle HA table rule space is represented by C_n. The set S represents these VNF servers and binary variable $h_{s,n} \in \{0, 1\}$ indicates whether server $s \in S$ is attached to FE $n \in N$.

VNFs. Different types of VNFs can be provided in a network. We assume that VNFs have been deployed on commodity servers located within the network. Let set $P = (P_{\mathrm{AS}}, P_{\mathrm{AL}}, P_{\mathrm{UA}})$ represent possible VNF types, where P_{AS} is the set of VNF types that require state connection and can only be accelerated in the FE node connected to its server, P_{AL} is the set of VNF types that are stateless and can be accelerated in any FE node, P_{UA} is the set of VNF types that cannot be accelerated in any FE node. Let the binary variable $h_{s,p} \in \{0, 1\}$ represent if VNF type p has been deployed on server s, and C_p represent the rule space requirement of VNF p if p is mapped into hardware.

Traffic Requests. We assume that there are several requests for setting up paths for different kinds of traffic. A traffic request is represented by a 4-tuple $t = < u^t, v^t, \Psi^t, \beta^t >$, where $u^t, v^t \in N$ denote the ingress and egress nodes, respectively, β^t is the volume of traffic to be processed (occupied bandwidth) and Ψ^t represents the ordered VNF sequence the traffic must pass through (for example, the FW-IDS-Proxy). For simplicity, a specific NF instance cannot be commonly used to handle two or more traffic requests because of the different traffic processing demand.

The VNF-HA mapping and traffic routing problem can be described as follows: given K traffic requests ($t_1, t_2, \ldots, t_n$), calculate the binary variable $h_{s,n,p,t} \in \{0, 1\}$ that represents whether the VNF p deployed in server s is to be accelerated in node n for traffic request t, while achieving the total optimization goals. Generally, the optimization goal is to minimize the overall operation cost by (i) mapping VNFs that need acceleration to the optimal FEs and (ii) installing the optimal routing path for each traffic request, while satisfying the capacity constraints and respecting the ordered NF sequence.

The operation cost consists mainly of the traffic forwarding cost: it represents the bandwidth costs that are used for transit links.

$$\text{Minimize} \quad \mathbb{Z} = \sum_{l=(u,v)} \sum_{j=1}^{K} \mathrm{y}_{l,j} \times \beta^{t_j}, u, v \in N, \quad (3)$$

Constraints:

$$y_{l,j} \in \{0, 1\}; \quad (4)$$

$$\sum_{j=1}^{K} y_{l,j} \times \beta^{t_j} \leq \beta_l, \quad (5)$$

$$\sum_{s \in S} \sum_{p \in \{P_{\mathrm{AS}}, P_{\mathrm{AL}}\}} \sum_{j=1}^{K} h_{s,n,p,t_j} \times h_{s,n} \times h_{s,p} \times C_p \leq C_n, n \in N, \quad (6)$$

$$\text{if } \Big\{(p \in P_{\mathrm{AS}}) \& (h_{s,n} \times h_{s,p} = 0)\Big\}, \text{then } h_{s,n,p,t_j} = 0, \quad (7)$$

$$\text{if } p \notin \{P_{\mathrm{AS}}, P_{\mathrm{AL}}\}, \text{then } h_{s,n,p,\mathrm{t_j}} = 0, \quad (8)$$

where the $\{0, 1\}$ variable $y_{l,j}$ indicates whether link l serves the traffic of the jth traffic request. Inequality (5)

https://doi.org/10.4218/etrij.17.0117.0174

shows the link bandwidth limitation, and (6) is the rule space limitation for each FE node, which means that the HA rules deployed on each FE should not exceed the idle HA table rule space of the FE. Equations (7) and (8) indicate the HA location limitation of the stateful VNFs and VNFs that cannot be accelerated in hardware.

This model belongs to integer linear programming, which can be solved by several classical heuristic algorithms, and the optimal algorithm is left for further study.

V. Evaluation

In this section, we first propose the prototype implementation of the HSN structure (shown in Fig. 2), which demonstrates that HSN can be implemented by simply using existing tools. Second, we verify the resource costs and processing performance of HSN.

1. SDN-Based Prototype

We implemented the HSN prototype using the NetFPGA-10G platform as the FE nodes and an SDN controller as the FE controller and HAM, as shown in Fig. 6.

(1) FE. Each NetFPGA-10G node has one FW table and three HA tables in hardware by simply modifying the OpenFlow1.3 abstraction: a classifier is added after the parser, and a match selector is added into each HA table. Each HA table has 64-bit fields for matching, and the match fields are selected by the match selector; while the FW table only performs the forwarding function, matching 5-tuple header fields (including port, IP_src, IP_dst, TCP_src, and TCP_dst). In the NetFPGA software, an OpenFlow agent [18] is deployed to connect with the FE controller and HAM also with a simple modification: match selector configurations of the HA tables are added during the mapping from the software flow table to the hardware.

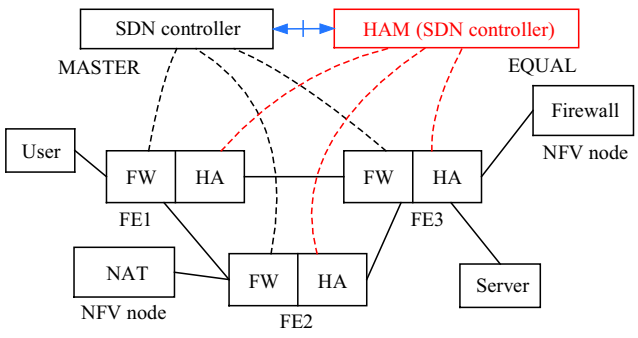

Fig. 6. HSN prototype.

(2) HAM. To control multiple FEs, we leverage the multi-controller mechanism of OpenFlow 1.3, in which there can be one MASTER controller and several EQUAL controllers. Initially, the multiple-controller mechanism is proposed to improve reliability, as the switch can continue to operate in OpenFlow mode if one controller or controller connection fails. In our prototype, this mechanism provides an exact solution for the HAM: the MASTER controller plays the role of network state collector, handling network-related events and ensuring network connectivity as well as traffic steering; the EQUAL controller plays the role of the HAM, communicating with VNFs and configuring the HA tables of each FE node. When OpenFlow operation is initiated, each FE can connect to all the controllers it is configured with and try to maintain connectivity with all of them concurrently. Many controllers may send controller-to-switch commands to the switch, the reply or error messages related to those commands must only be sent on the controller connection associated with that command.

2. FE Resource Analysis

The FE node was built on a NetFPGA-10G card connected with a common server (2.5 GHz 64-bit CPU and 16 GB DDR2 RAM) for software OpenFlow agent deployment. Given limited hardware resources, the transfer unit contains four 10 Gbps physical ports connected to Ethernet and four CPU virtual ports in direct memory access (DMA) connected to the server. As for the match tables, we used TCAM and a hash table to store and match functional rules. This implementation was built based on the OpenFlow switch project on NetFPGA [19].

First, we evaluated the optimal volume of each HA table. Because of the limited resources in NetFPGA, a TCAM HA table usually can only implement dozens of rules, which may not be representative of commodity hardware devices. Therefore, we assume that the depth of each HA table can meet the VNF requirements and only consider the optimal match width. Generally, the match fields of each VNF are limited, thus we assumed there are two VNFs to be mapped in an FE (considering the limited hardware resources of NetFPGA) and randomly selected three match fields from the OpenFlow 12-tuple as the required match fields of each VNF. Then, we made the statistics of the average resource costs and table utilization under different HA table match width, as shown in Table 1.

From Table 1 we can see that the resource cost for a 16-bit match width is the highest, because there are more tables required as well as the additional action processor in each table. Compared to a 32-bit match width, the table

Table 1. HA table resource cost of different match widths.

Width (bits)	Slice	BRAM	Table Utilization (%)
16	26,388	60	95.43
32	24,232	48	87.65
64	14,147	25	83.35
96	12,939	23	73.75
128	16,873	26	68.97

Table 2. Resource cost of the OpenFlow and HSN data path.

Design	Slice	BRAM	Utilization (%)
OpenFlow1.3	33,816	256	90.3
HSN	35,454	263	94.7

utilization for a 64-bit match width is 4.3% lower, but the resource cost is lower by 41.6%. Moreover, compared to a 96-bit match width, the table utilization in 64-bit match width increases by 10% while the resource cost only increases by 0.8%. The table utilization of HA table with 128 bits is the lowest, which means there are more resource wasted if the width is too large. In fact, the optimal match width is the balance between resource cost and table utilization, and the experimental results show that 64-bit of HA table match width is a good compromise.

Second, we evaluated the resource cost of HSN by comparing the mapping reports between OpenFlow1.3 and the HSN data path built on NetFPGA-10G. To make a fair comparison, these two data planes shared the same match table volume (a total match width of 296 bits) and header vector length (512 bits): the HSN data plane contained one FW table with 5-tuple match fields and three HA tables with a 64-bit match width, while the OpenFlow data plane contained four flow tables. Total slice and BRAM counts are shown in Table 2.

From Table 2, we can see that although HSN uses additional resources to support classifier and VNF-defined match selectors, these two data planes nearly have the same hardware resource costs. The experimental results show that HSN can achieve high flexibility while maintaining a low cost for HA table programmability.

3. Throughput Evaluation

In our implementation, we used NOX [20] as an SDN controller, and this controller ran on an Ubuntu 12.04 system in a Lenovo computer with an Intel® Core™2 i7-3770 (3.50 GHz) CPU. Because of the lack of VNFM, to simplify the communication between VNF and HAM, we used OVS instead of VMs to build two VNFs: a stateful Firewall and a stateless NAT. We used Spirent TestCenter to generate and send original packets with lengths of 256 bytes in our testbed environment shown in Fig. 6.

First, we tested the performance of stateless NAT using and not using HA, respectively. For one node (FE2) shown in Fig. 7(a), because of the parallel hardware processing, the forwarding latency of software VNF was more than ten times that of the hardware VNF, which means that VNFs can achieve a much better performance via HSN HA. For the forwarding latency between the user and server shown in Fig. 7(b), the forwarding latency of non-HA was also far larger than that of HSN. In HSN architecture, most of the packets can be processed according to the HA table in local switches without being sent to the software VNF to match rules and perform actions. Thus, the performance was improved substantially in the HSN architecture.

Second, we compared the performance of HSN with a controller-based HA (shown in Fig. 1(b)) that integrates the two components into one controller. For simplicity, the VNF–HA mapping was not used. As shown in Fig. 8, the

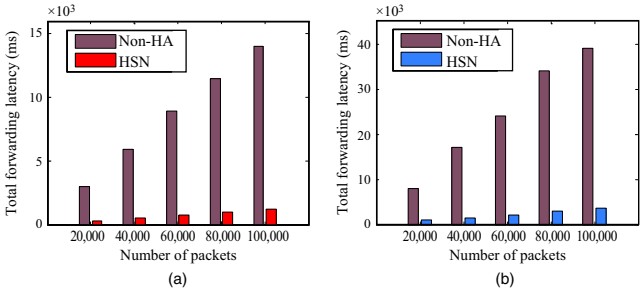

Fig. 7. Performance of HA and non-HA: (a) forwarding latency of one node; (b) forwarding latency between user and server.

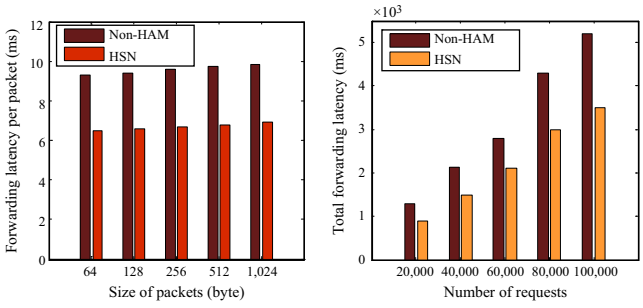

Fig. 8. Performance of HA with and without HAM: forwarding latency between the user and server.

https://doi.org/10.4218/etrij.17.0117.0174

average forwarding latency of the stateful Firewall in HSN was reduced by about 30% compared with that of the controller-based HA, because VNFs built on the controller involve additional flows sent to the controller for processing, which increases the FE-to-controller load and path setup latency.

VI. Conclusion

In this paper, we proposed a new paradigm for NFV HA in the SDN/NFV architecture, which can effectively reduce the FE controller load while making full use of the network data plane processing ability by separating the processing and control of SDN and NFV. The corresponding NFC mechanism was also discussed, including the VNF-HA mapping process and mathematical analysis of VNF-HA mapping and traffic routing problem. The prototype built on NetFPGA-10G and SDN controller demonstrated that the processing performance can be greatly improved with only small modifications of the traditional SDN/NFV architecture. Our future work will mainly focus on NFV HA via FEs that enable stateful packet processing, as well as the orchestration algorithms of these acceleration architectures.

References

[1] ONF, *OpenFlow-Enabled SDN and Network Functions Virtualization*, white paper, 2014.

[2] H. Song, "Protocol-Oblivious Forwarding: Unleash the Power of SDN through a Future-proof Forwarding Plane," *HotSDN'13*, Hong Kong, China, Aug. 16, 2013, pp. 127–132.

[3] P. Bosshart et al., "P4: Programming Protocol-Independent Packet Processors," *ACM Comput. Commun. Rev.*, vol. 44, no. 4, Apr. 2013, pp. 87–95.

[4] Z. Bronstein et al., "Uniform Handling and Abstraction of NFV Hardware Accelerators," *IEEE Netw.*, vol. 29, no. 3, May–June 2015, pp. 22–29.

[5] J. Bi et al., "Supporting Virtualized Network Functions with Stateful Data Plane Abstraction," *IEEE Netw.*, vol. 30, no. 3, May–June 2016, pp. 40–45.

[6] M.T. Arashloo et al., "SNAP: Stateful Network-Wide Abstractions for Packet Processing," *Conf. ACM SIGCOMM'16*, Florianopolis, Brazil, Aug. 22–26, 2016, pp. 29–43.

[7] A. Dwaraki and T. Wolf, "Adaptive Service-Chain Routing for Virtual Network Functions in Software-Defined Networks," *Conf. ACM HotMiddlebox*, Florianopolis, Brazil, Aug. 22–26, 2016, pp. 32–37.

[8] J. Matias et al., "Toward an SDN-Enabled NFV Architecture," *IEEE Commun. Mag.*, vol. 53, no. 4, Apr. 2015, pp. 187–193.

[9] X. Ge et al., "OpenANFV: Accelerating Network Function Virtualization with a Consolidated Framework in Openstack," *ACM Comput. Commun. Rev.*, vol. 44, no. 4, Oct. 2015, pp. 353–354.

[10] B. Li et al., "ClickNP: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware," *Conf. ACM SIGCOMM'16*, Florianopolis, Brazil, Aug. 22–26, 2016, pp. 1–14.

[11] A. Bremler-Barr et al., "Deep Packet Inspection as a Service," *Conf. ACM CoNEXT*, Sydney, Australia, Dec. 2–5, 2014, pp. 271–282.

[12] A. Bremler-Barr et al., "OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions," *Conf. ACM SIGCOMM*, Florianopolis, Brazil, Aug. 22–26, 2016, pp. 511–524.

[13] ETSI, *Network Functions Virtualization (NFV); Infrastructure Overview*, ETSI ISG for NFV, 2015, Accessed Nov. 15, 2016. http://www.etsi.org/deliver/etsi_gs/NFV-INF/001_099/001/01.01.01_60/gs_nfv-inf001v010101p.pdf

[14] P. Bosshart et al., "Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN," *ACM Comput. Commun. Rev.*, vol. 43, no. 4, Oct. 2013, pp. 99–110.

[15] S.K. Fayazbakhsh et al., "Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions Using Flowtags," *NSDI'14*, Seattle, WA, USA, Apr. 2–4, 2014, pp. 533–546.

[16] Z.A. Qazi et al., "SIMPLE-fying Middlebox Policy Enforcement Using SDN," *ACM Comput. Commun. Rev.*, vol. 43, no. 4, Oct. 2013, pp. 27–38.

[17] L. Jose et al., "Compiling Packet Programs to Reconfigurable Switches," *NSDI'15*, Oakland, CA, USA, May 4–6, 2015, pp. 103–115.

[18] G. Gibb, *OpenFlow Agent for NetFPGA-10G*, Stanford University, 2011, Accessed Nov. 3, 2016. http://git://github.com/eastzone/openflow.git

[19] T. Yabe, *OpenFlow implementation on NetFPGA-10G Design Document*, Stanford University, 2011, Accessed Nov. 3, 2016. https://github.com/NetFPGA/NetFPGA-public/wiki

[20] Stanford University, *NOX Controller*, Github, 2012, Accessed Nov. 4, 2016. https://github.com/noxrepo/nox

Tong Duan received his BS degree in electronic science and technology from Tsinghua University, Beijing, China, in 2013 and his MS degree in information and communication engineering from the National Digital Switching System and Technology Research Center (NDSC), Zhengzhou, China, in 2016. He is currently pursuing his PhD in communication and information systems, NDSC. His research interests include programmable network data planes and network function virtualization.

Julong Lan was born in 1962. He is a professor in information and communication engineering and the chief engineer of the National Digital Switching System and Technology Research Center, Zhengzhou, China. His research contributions encompass aspects of information theory, network communication, and future network architecture.

Yuxiang Hu received his PhD in information and communication engineering from the National Digital Switching System and Technology Research Center (NDSC), Zhengzhou, China, in 2011. He is currently an associate professor at NDSC. His research mainly focuses on future network architecture.

Penghao Sun received his BS and MS degrees in information and communication engineering from the National Digital Switching System and Technology Research Center (NDSC), Zhengzhou, China, in 2017. He is currently pursuing his PhD in communication and information systems, NDSC. His research interests are network packet classification and processing.

https://doi.org/10.4218/etrij.17.0117.0174