Supporting Virtualized Network Functions with Stateful Data Plane Abstraction

Jun Bi, Shuyong Zhu, Chen Sun, Guang Yao, and Hongxin Hu

Abstract

A recent trend known as NFV attempts to replace dedicated hardware appliances with generic compute, storage, and network resources. NFV based on SDN can extend programmability and flexibility features of advanced network functions. Many of these network functions, however, need concrete state information to effectively process network flows. However, OpenFlow only provides a simple "match-action" paradigm and lacks the function of stateful processing in the SDN data plane, which limits its support of advanced network functions. Heavily relying on SDN controllers for all state maintenance incurs both scalability and performance issues. In this article, we propose a novel SDPA for the SDN data plane. A co-processing unit, the forwarding processor, is designed for SDN switches to manage state information through new instructions and state tables. We demonstrate the practicality and feasibility of our approach through a proof-of-concept implementation of VNFs, a stateful firewall based on SDPA. Experimental results show that VNFs which need to process state information can be supported well by the SDPA architecture, and the forwarding efficiency is improved.

Virtualized network functions (VNFs) are the basic units of network function virtualization (NFV). Currently, the most common implementation mode of VNF is running each function on a standalone virtual machine (named VM-VNF), which allows great flexibility. However, the flexibility comes with considerable compromises. First, VM carried functions may introduce significant performance issues (e.g., latency, throughput). Especially if VNFs should process data packets (e.g., firewall, network address translation), these problems will be worse. However, due to the VNFs' lack of a unified paradigm, it is very hard, if not impossible, to improve the performance of all the VNFs through dedicated optimization or hardware acceleration. Second, the NFV management and orchestration plane (MOP) actually does not have an explicit view into the VNFs for what they really execute at runtime. The VNFs can conflict with each other and may introduce errors. However, the MOP has a lack of mechanisms to effectively detect and resolve the conflicts and errors [1].

Software defined networking (SDN) can be used to implement VNFs. Whenever a VNF is implemented based on SDN, the VNF could be composed by a virtualized data plane device and an SDN controller application. The controller application generates rules to apply to the data traffic, and the data plane device processes packets based on the rules. Specifically, implementing VNF with SDN (named SD-VNF) can make up for the drawbacks of VM-VNF. First, the rules on the virtualized data plane device can be implemented by specially optimized flow tables or even hardware switches. Hence, the data plane performance can be improved. Second, the centralized SDN controller is capable of checking errors or conflicts of various VNFs at runtime.

In theory, through leveraging the capability of the controller, the SD-VNF can implement any function. However, the majority of network functions, other than the functions' generating static rules, require invoking the controllers frequently. Hence, the controllers and the channel between the data plane and the control plane will become the bottlenecks of the functions. For example, even only when implementing a stateful firewall, the packets should be sent to the controller to keep the state of TCP connection or UDP pseudo connection. We believe this problem originates from the limitation of the expression capability of the SDN data plane. As a representative technique of SDN, OpenFlow [2] introduces a "match-action" paradigm for the SDN data plane where programmers could specify a flow through a header matching rule along with processing actions applied to matched packets.

OpenFlow's simple "match-action" abstraction brings great challenges to support NFV, which requires advanced packet handling. First, the OpenFlow data plane only provides limited support for stateful packet processing and is unable to monitor flow states without the involvement of the controller [3]. Heavily relying on the controller to maintain all packet states could give rise to both scalability and performance issues due to the associated processing delay and the control channel bottleneck between the controller and switches [4]. Second, OpenFlow targets fixed-function switches that recognize a predetermined set of header fields and processes packets using a small set of predefined actions. The header fields and actions cannot be extended flexibly to meet diverse application requirements. The limited expressivity of OpenFlow compromises the programmability and capability of the SDN data plane [5, 6].

To address the above-mentioned challenges and requirements, we introduce an innovative stateful data plane abstraction (SDPA) to enable stateful processing in the SDN data plane. We try to close the gap between the requirement and the capability, while keeping the advantages of the current SDN data plane. We found, however, that a stateful data plane is required to cover all the functions. In contrast to the simple

Jun Bi, Shuyong Zhu, Chen Sun, Guang Yao are with Tsinghua University; Hongxin Hu is with Clemson University.
The corresponding author is Jun Bi.

0890-8044/16/$25.00 © 2016 IEEE

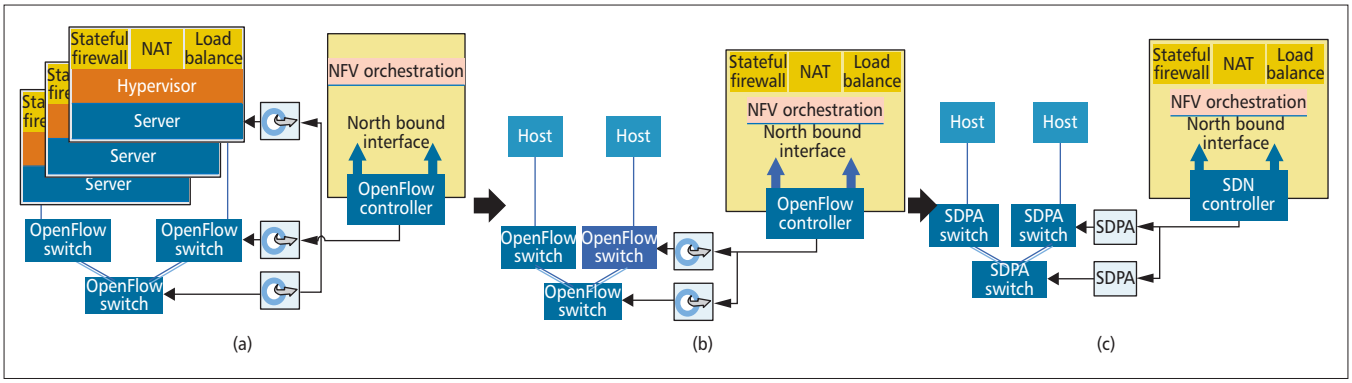

Figure 1. The architectures of how SDN supports NFV: a) OpenFlow-enabled SDN benefit for NFV; b) OpenFlow-enabled SDN-based NFV; c) SDPA-enabled SDN-based NFV.

"match-action" paradigm of OpenFlow, we propose a new "match-state-action" paradigm for the SDN data plane. In this paradigm, state information can be maintained in the SDN data plane without heavy involvement of SDN controllers. In particular, we add a co-processing unit named forwarding processor (FP) to SDN switches, and design instructions and flow tables in the FP to manipulate state information in the SDN data plane.

SDN envisions intelligent and centralized controllers governing the forwarding behavior of dumb and low-cost switches. However, are the dumb switches a strategic choice in reality? Some dynamic applications, which only involve local states inside single links or switches, are unnecessarily centralized for easy management and programmability. Indeed, some level of control logic in switches could be handled just based on local states inside the device itself, which can efficiently offload the centralized controller from decisions.

The article makes the following contributions:

- We propose the novel stateful data plane abstraction (SDPA) to support NFV. This architecture proposes a new "match-state-action" paradigm to support various functions that need to process state information in the SDN data plane.
- We design a co-processing unit for SDN switches along with instructions and state tables to support stateful processing in the SDN data plane. Through adding intelligence to SDN switches, the programmability and flexibility of the SDN data plane can be greatly enhanced.
- We implement a typical network function, a stateful firewall, based on the proposed SDPA architecture, and demonstrate how our approach can effectively support VNFs.

Stateful Data Plane Abstraction

Figure 1 depicts several architectures for SDN support of NFV. Figure 1a is derived from [7]. In this architecture, virtualized functions run on local servers while only NFV orchestration is implemented as an app running on the controller. In this architecture, OpenFlow-enabled SDN brings benefits to NFV.

In Fig. 1b, most of the network functions are running on top of the controller. The switches are OpenFlow switches. The controller communicates with switches through OpenFlow protocol. This model attempts to support NFV using the SDN architecture. We call it OpenFlow-enabled SDN-based NFV. However, some network functions, other than the functions generating static rules, require invoking the controllers frequently. Hence, the controllers and the channel between the data plane and the control plane will become the bottlenecks. This model may incur performance issues such as forwarding latency and throughput. Another problem with the architecture of Fig. 1b is the limited functionality that OpenFlow provides.

Figure 1c shows how SDPA architecture supports NFV. We call it SPDA-enabled SDN-based NFV. In this architecture, we use SDPA switches that can process state information in the data plane. Each function is divided into two parts. One part is a stateful processing unit inside SDPA switches. The other is the associated function running on the controller. This architecture is compatible with OpenFlow or SDN-enabled switches. And the bottlenecks of controllers and the channel between the data plane and the control plane can be avoided in this architecture. Some special network functions that cannot easily be implemented in SDPA switches can still be placed on local servers connected to SDPA switches, similar to the mechanism in Fig. 1a.

Concretely, we design a co-processing unit in SDN switches, the FP, which can be implemented by CPU, NPU, and so on, as shown in Fig. 2. Each stateful processing unit in the data plane is composed of a state table (ST), a state transition policy table, and a flow table. When implementing stateful network applications such as stateful firewalls, input packets are processed according to related state information. The ST is used to maintain the state of each TCP connection or UDP pseudo connection, and it is dynamically updated according to the coming packets (e.g., the TCP flag) or internal/external events. The transition of TCP connection state can be described using a finite state machine. And the state transition policy table is used to keep the finite state machine of TCP connection. Through matching the state transition policy table, the next state of the TCP connection and corresponding actions are determined.

All the stateful processing units are composed together in parallel or sequentially according to application requirements. Through extended OpenFlow instructions, flows or packets are directed from the OpenFlow pipeline to the FP. The FP realizes more complex processing of flows or packets through instructions. We design STs for the FP, in which the FP maintains the associated state of flows or packets.

A New Paradigm for the SDN Data Plane

In SDN architecture, some VNFs need to process state information in the data plane. OpenFlow's "match-action" paradigm is simple and capable enough to support many data plane functions, but provides limited support for stateful processing due to the lack of state-related modules in the pipeline of the OpenFlow date plane. In essence, the limited "match-action" paradigm seems to be an involuntary outcome of being amenable to high-performance low-cost implementations, without taking into account a rich set of complicated network services, such as stateful firewalls, load balancing, FTP, intrustion detection/prevention system (IDS/IPS), NAT, and medium access control (MAC) learning.

We propose a new "match-state-action" paradigm for the SDN data plane, as shown in Fig. 3. In this paradigm, we add STs and state operating instructions to enable stateful processing in the SDN data plane. The STs are used to keep state

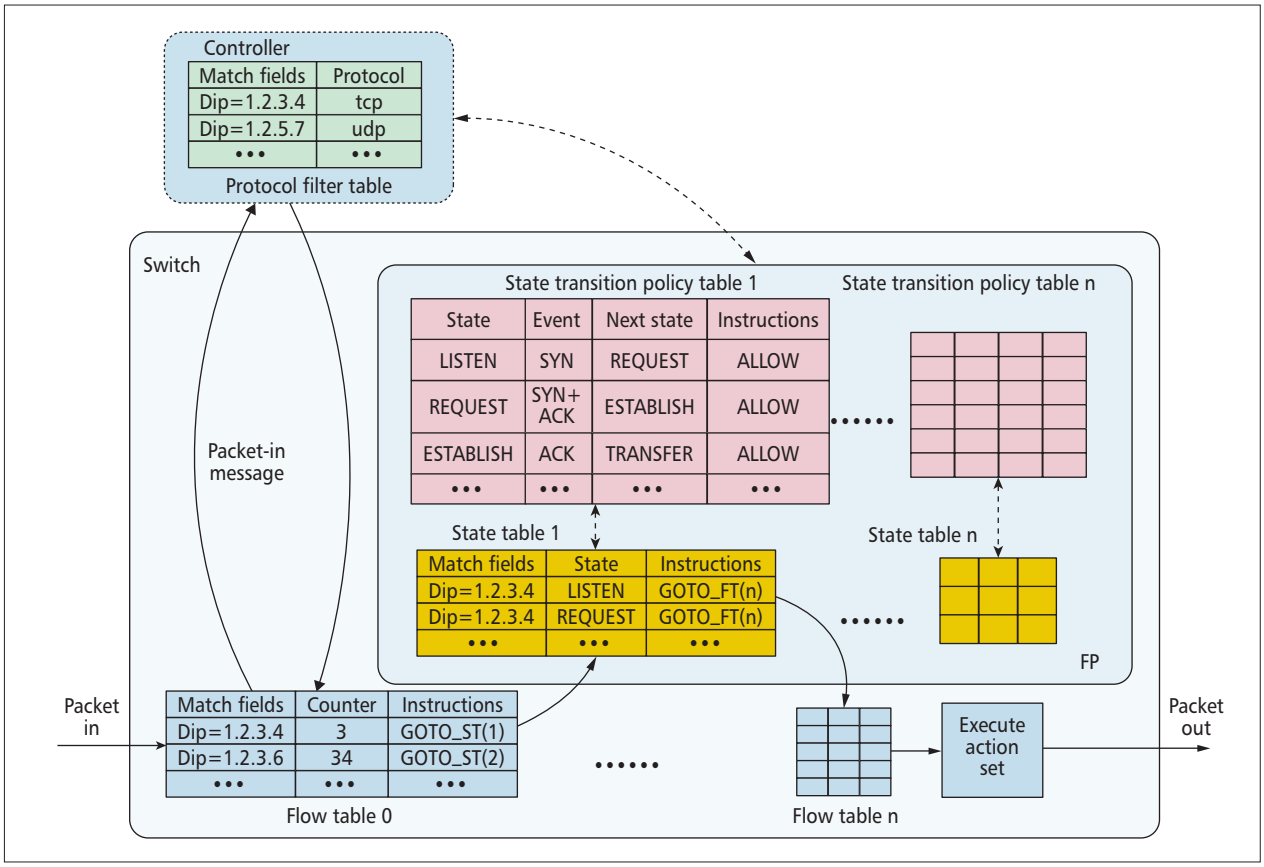

Figure 2. Detailed SDPA architecture.

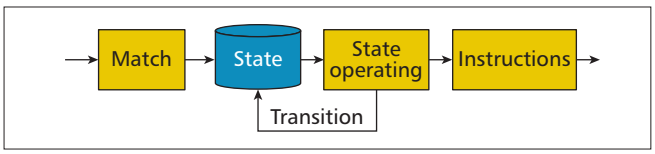

Figure 3. SDPA paradigm.

information of packets, flows, or connection. The state operating instructions are used to maintain state information. With this new paradigm, state processing can be programmed by SDN applications, and the state information can be maintained in the SDN data plane. Thus, based on this paradigm, stateful processing can be efficiently supported in the data plane without conveying all packets to the controller for state information maintenance.

Forwarding Processor

An FP maintains the state of flows or packets. Also, it can modify the metadata of packets, and initiate or delete ST entries asynchronously. An FP can receive and react to incoming events, such as configuration change, state change, or just packets, from both the controller and the switch. The controller can initiate, read, and modify the ST or state transition table in the FP. The events mean changes of network state, such as a new coming packet, connection interruption, network congestion, and so on.

We add a GOTO_ST(n) instruction in the data plane, which is used to direct packets from an OpenFlow pipeline to ST n in an FP. The packet is directed from the FP back to flow table m through the instruction GOTO_FT(m). We design instructions for stateful processing in an FP. These instructions can be flexibly extended to meet application requirements in the data plane. The instructions can be divided into the following categories:

- *Control instructions:* They are used to direct packets transferring between the controller, flow tables, and FP, including GOTO_ST(n) and GOTO_FT(m).
- *Processing instructions:* They are used for FP to process flows or packets.
- *State operating instructions:* They are used to operate the ST.
- *Arithmetic instructions:* They are used to perform arithmetic operations.
- *Logical instructions:* They are used to perform logical operations.

The controller and FP communicate with each other through an extended OpenFlow protocol. It is mainly used for the operation of state information in the data plane, such as initialization of the ST and state transition table. The controller has full control of the FP. We design two new message types, *controller-to-FP messages* and *asynchronous messages*. Each of them contains multiple sub-types.

State Table

State tables are used to maintain state information in the SDN data plane. Since different protocols may need to maintain different state information, each protocol has a corresponding ST. State tables are initiated by the controller. When an application requires stateful processing, the controller instructs the FP to initiate corresponding STs through an extended instruction INIT. The controller tells the FP explicitly which domains the ST should have. State tables are dynamically loaded. The match fields of STs may be different for various applications. One stateful application only has one ST and one state transition policy table. The entries of an ST could be effectively limited by the expiry mechanism.

The state information is updated according to incoming packets or internal/external events, and maintained in the data plane. The state information can also be uploaded to the controller through the asynchronous messages so that the control-

ler can keep the *global* state information of the network. When the state information is updated in the FP, it can be sent to the controller to retain consistency. Figure 2 shows examples of state tables. The "match fields" domain in an ST refers to the match fields of packets. It is flexible and extensible. For example, it can store connections possibly represented by both source and destination addresses. The "state" domain in an ST is used to record the state information of flows or packets. And the "instructions" domain is utilized to record associated processing instructions to process packets and update the states. Those instructions can be divided into *state operating* instructions and *packet processing* instructions.

State Transition Table

We design a state transition table to support the specification of state update policies with respect to a specific connection-oriented protocol. Each ST should be accompanied by a state transition table. The state transition policy tables may also be different for different applications. The entries of a state transition policy table are very limited. Taking a stateful firewall as an example, the state transition policy table only has a dozen entries.

A state transition table specifies the transition policies indicating how the states transfer according to the specific protocol. A state transition table contains four different domains, including *state*, *event*, and *next state*. State transition tables are issued to an FP by the controller.

Protocol Filter Table

As a general architecture, SDPA can support a variety of applications and protocols. Since different applications should maintain different state, each protocol used by the applications should have a specific ST. The protocol filter table should be established in advance in the controller to determine to which protocol the connection belongs. An example of protocol filter table is shown in Fig. 2 where the match fields domain refers to the matching domains, and the protocol domain refers to which protocol the filtered packet is using.

When a matching in the protocol filter table succeeds, a corresponding flow entry is issued by the controller carrying the extended instruction GOTO_ST(*n*). The parameter *n* refers to the ST *ID* to which the packet should be sent. At the same time, corresponding ST domain information and a state transition table are issued to the FP for stateful processing in the data plane.

SDN Switch Architecture Supporting SDPA

We design an SDN switch architecture supporting SDPA as shown in Fig. 4. We add the FP and ST to the SDN switch architecture to maintain the state information in the data plane. We also add a policy module, which is used to adjust the processing policies. This module includes the state transition table discussed above.

The new architecture consists of the following functional modules.

Network Interface: Directly connected to the physical layer. Its main functions include receiving/sending packets and packet processing. It works in the physical layer and the link layer.

Forwarding Engine: Responsible for determining the packet forwarding paths. It parses the received packet headers and looks up the forwarding table to obtain the destination ports for the forwarding operation.

Forwarding Processor: It interacts with the controller and is responsible for the maintenance and management of state information in the data plane.

Forwarding Table: Plays the role of connecting the entire system. It can be updated according to the information issued by the controller and returns associated forwarding instructions to the forwarding engine.

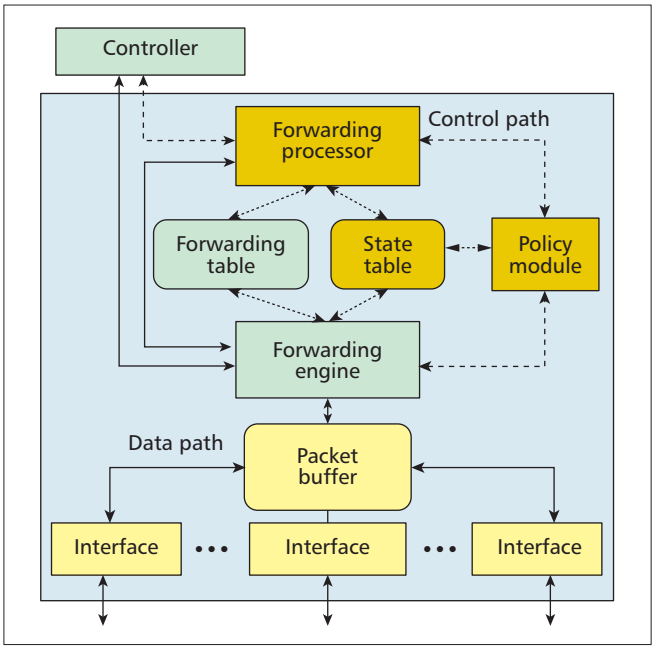

Figure 4. SDN switch architecture supporting SDPA.

State Table: It is used to maintain state information during the processing procedure in the data plane.

Policy Module: Used to adjust and control the processing policies, such as the state transition policy and packet processing policy, of the switch. The policies are issued by the controller.

Use Case and Evaluations

To demonstrate the feasibility and efficiency of our approach, we implemented a stateful firewall, a typical VNF, based on the SDPA architecture and evaluated its performance.

Implementation of a Stateful Firewall

In our implementation, we extended Open vSwitch (OVS) [8] to support FP and used NOX [9] as the SDN controller, on which we developed a VNF, a stateful firewall. All our experiments were performed in the Ubuntu 12.04 system running on a Dell OPTIPLEX 780 computer. The CPU of this computer is Intel® Core™ 2 Duo Processor E7500 (2.93 GHz), and the internal memory is 3.21 GB. The network card is an Intel 10 Gigabit Network Connection. We used IXIA [10] to generate and send original packets in our testbed environment.

We implemented the stateful firewall application based on the SDPA architecture, as shown in Fig. 1c, where an FP is used to maintain the state of TCP connections and UDP pseudo connections. The ST resides in the FP to record state information. The match fields domain consists of *SIP*, *SPORT*, *protocol*, *DIP*, and *DPORT*. The state domain contains *Connection state*, *Sequence number*, *Acknowledge number*, *Idle timeout*, and *Hard timeout*. The instructions domain includes *state operating instructions* and *packet processing instructions.*

Performance of Stateful Firewalls in SDPA Architecture against Stateful Firewalls in Traditional SDN Architecture: We conducted a contrast experiment to evaluate the efficiency of SDPA. We needed to evaluate the performance of processing states in switches based on the SDPA architecture against processing states in the controller based on the traditional SDN architecture. We also developed a stateful firewall application based on the traditional SDN architecture, where the state information is maintained in the controller as shown in Fig. 1b.

We tested the forwarding latency and the throughput in our experiment. As we can see from our experiment results, when

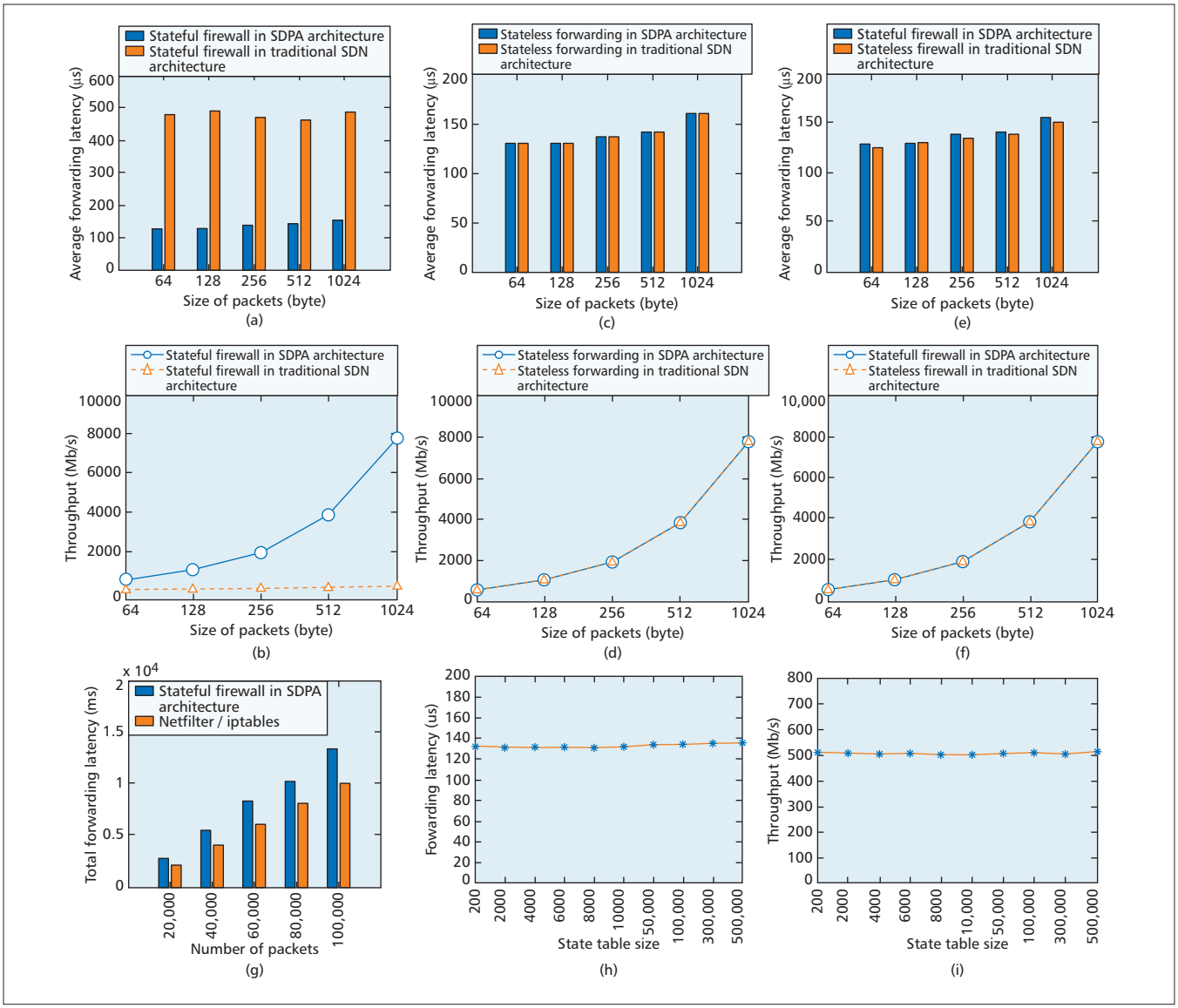

Figure 5. Performance evaluation and comparison: a) forwarding latency; c) forwarding latency; e) forwarding latency; b) throughput; d) throughput; f) throughput; g) total forwarding latency; h) forwarding latency; and i) throughput.

packet size ranges from 64 to 1024 bytes, the average forwarding latency reduces more efficiently in SDPA architecture than that in the transitional SDN architecture as shown in Fig. 5a. In SDPA architecture, most of the packets can be processed according to the ST in local switches without being sent to the controller to match the firewall rules. Thus, the throughput increases a lot in SDPA architecture, as shown in Fig. 5b.

Performance of Stateless Forwarding in SDPA Architecture vs. Traditional SDN Architecture: Since the SDPA architecture is fully compatible with OpenFlow, SDPA can also support stateless processing just like OpenFlow. While performing stateless processing in the data plane, the average forwarding latency in the SDPA architecture is almost the same as that in the traditional SDN architecture, as shown in Fig. 5c. And the throughput in the SDPA architecture is almost the same as that in the traditional SDN architecture depicted in Fig. 5d.

Performance of Stateful Firewalls in SDPA Architecture vs. Stateless Firewalls in Traditional SDN Architecture: We compared our stateful firewall in the SDPA architecture with a *stateless* firewall in the traditional SDN architecture. Regarding the stateless firewall in traditional SDN architecture, only the first packet of a flow is sent to the controller to match firewall rules. Then the controller issues a new flow entry to the flow table. The subsequent packets of the flow are in turn directly matched against the flow table. As can be seen in Fig. 5e, the average forwarding latency of the stateful firewall in the SDPA architecture is slightly increased. The processing overhead is acceptable, and the throughput rate is nearly unchanged as shown in Fig. 5f.

Performance of Stateful Firewall in SDPA Architecture vs Stateful Firewall Netfilter/iptables: Netfilter/iptables [11] is a user-space application program that allows a system administrator to configure the tables provided by the Linux kernel firewall, and the chains and rules it stores. Nevertheless, our stateful firewall is an application running on top of a controller to enable effective state information processing in SDN-based networks. We selected a 10 gigabit network card and used a 64-byte packet to conduct our experiment. As shown in Fig. 5g, the total forwarding latency of stateful firewalls in the SDPA architecture is slightly higher than that of netfilter/iptables. The packet loss rates of both kinds of firewalls are almost the same.

Testing the Scalability of State Tables

We performed a test on the scalability of STs and the influence of forwarding efficiency under different sizes of STs. We also used 64-byte packets to conduct our experiment. As the

ST size increases, the forwarding efficiency does not noticeably deteriorate. Since the ST is implemented based on SRAM in our experiment, the size of the ST can still be increased theoretically. As shown in Fig. 5h, when the size of an ST increases from 200 to 500,000, the network forwarding latency does not increase significantly. And the network throughout shows almost no change, as shown in Fig. 5i. This indicates that maintaining state in the data plane has little impact on forwarding latency and throughput.

Related Work

Some research efforts have recently been devoted to extending the OpenFlow data plane abstraction [12–14]. Bosshart *et al.* [12] pointed out that the rigid table structure of current hardware switches limits the scalability of OpenFlow packet processing to match on a fixed-set of fields and to a small set of actions. By comparison, we strive to enhance the programmability of the data plane by adding a co-processing unit in SDN switches. In addition, Bianchi *et al.* [13] proposed a new abstraction to formally describe a desired stateful processing of flows inside the SDN data plane based on extended finite state machines. Moshref *et al.* [14] proposed flow-level state transitions as a new switch primitive for SDN. They just put forward a preliminary design, but did not provide concrete implementations and evaluations. In contrast, we present a detailed technical scheme for realizing our SDPA architecture in the SDN data plane to support NFV, and the relationships and interactions between the state tables and flow tables are articulated. We implemented a typical VNF, a stateful firewall, based on our SDPA architecture along with convincing experimental results.

Conclusion and Future Work

Software defined networking techniques can be used to implement VNF. However, OpenFlow-enabled SDN-based NFV still has performance issues due to the bottlenecks of the controller, and the channel between the control plane and the data plane. OpenFlow only provides a simple match-action paradigm and lacks the function of stateful processing for the SDN data plane, which limits its support for advanced network applications. In this article, we have put forward SDPA to support NFV. We have designed a co-processing unit, the FP, which can help manipulate states in the SDN data plane. The feasibility and efficiency of our approach are demonstrated through the implementation of a stateful firewall. Our implementation and evaluations showed that the SDPA architecture has the following advantages:

- VNFs that need to maintain state information can be supported well in the SDPA architecture, and the forwarding efficiency can be improved efficiently.
- The SDPA architecture is fully compatible with OpenFlow. Applications that do not need to maintain state information in the data plane can be fully supported as well without causing additional processing overhead.
- The performance of stateful processing in the SDPA architecture is close to that of stateless processing in the traditional SDN architecture.
- The SDPA architecture enhances the programmability and flexibility of the data plane significantly.

For future work, we will develop more network applications, such as DNS reflection attack defense, fast reroute, and NAT, based on the SDPA architecture to validate the versatility and availability of SDPA. We will also develop an SDPA hardware prototype system based on NetFPGA[15] and further evaluate the performance of SDPA architecture. Adding some intelligence into switches may increase their complexity. Thus, we will investigate an optimized solution to simplify our design.

Acknowledgment

The authors would like to acknowledge the support from the National High-Tech R&D Program ("863" Program) of China (No. 2013AA013505) and the National Science Foundation of China (No. 61472213 and No. 61303194).

References

[1] "Network Functions Virtualisation," updated white paper; https://portal.etsi.org/nfv/nfv_white_paper2.pdf
[2] N. McKeown *et al.*, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Comp. Commun. Rev.*, vol. 38, no. 2, 2008, pp. 69–74.
[3] H. Song, "Protocol-Oblivious Forwarding: Unleash the Power of SDN through a Future-Proof Forwarding Plane," *Proc. 2nd ACM SIGCOMM Wksp. Hot Topics in Software Defined Networking*, Aug. 2013, pp. 127–32.
[4] M. Jarschel *et al.*, "Modeling and Performance Evaluation of an Openflow Architecture," *Proc. 23rd Int'l. Teletraffic Congress*, Sept. 2011, pp. 1–7.
[5] P. Bosshart *et al.*, Programming Protocol-Independent Packet Processors, arXiv preprint arXiv:1312.1719, 2013.
[6] B. Anwer *et al.*, "A Slick Control Plane for Network Middleboxes," *Proc. 2nd ACM SIGCOMM Wksp. Hot Topics in Software Defined Networking*, ACM, Aug. 2013, pp. 147–48.
[7] ONF, "OpenFlow-Enabled SDN and Network Functions Virtualization," white paper, 2014.
[8] Open vSwitch: http://openvswitch.org/
[9] NOXRepo: http://www.noxrepo.org/
[10] Ixia: http://www.ixiacom.cn/
[11] netfilter/iptables project: http://www.netfilter.org/
[12] P. Bosshart *et al.*, "Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN," *Proc. ACM SIGCOMM 2013 Conf.*, Aug. 2013, pp. 99–110.
[13] G. Bianchi, "OpenState: Programming Platform-Independent Stateful Openflow Applications Inside the Switch," *ACM SIGCOMM Comp. Commun. Rev.*, vol. 44, no. 2, 2014, pp. 44–51.
[14] M. Moshref *et al.*, "Flow-Level State Transition as a New Switch Primitive for SDN," *Proc. 2014 ACM Conf. SIGCOMM*, Aug. 2014, pp. 377–78.
[15] NetFPGA: http://netfpga.org/

Biographies

JUN BI [SM] (junbi@tsinghua.edu.cn) received B.S., M.S., and Ph.D. degrees in computer science from Tsinghua University, China. He was a postdoctoral scholar at Bell Laboratories Research and a research scientist at Bell Labs. Currently he is a full professor and director of the Network Architecture Research Division, Institute for Network Sciences and Cyberspace at Tsinghua University, and a key member of the Tsinghua National Laboratory for Information Science and Technology. His research interests include Internet architecture and protocols. He has successfully led tens of government supported or international collaboration research projects, published more than 100 research papers and 20 Internet RFCs or drafts (four of them were approved), owns 20 innovation patents, and has received national science and technology advancement prizes. He is Co-Chair of the Asia Future Internet Forum (AsiaFI) Steering Group and Co-Founder of the China SDN Commission, and serves as Executive Chair. He has served as Co-Chair of Workshops/Tracks at INFOCOM, ICNP, Mobihoc, ICCCN, and so on, and has served on the Organization Committees or Technical Program Committees at SIGCOMM, ICNP, CoNEXT, SOSR/HotSDN, and so on. He is a Senior Member of ACM and a Distinguished Member of the China Computer Federation.

SHUYONG ZHU (zhu-sy11@mails.tsinghua.edu.cn) received B.S. and M.S. degrees in computer networks from the National University of Defense Technology, China. He is a Ph.D. student at the Institute for Network Sciences and Cyberspace, Tsinghua University. His research fields include Internet architecture, software-defined networking, and network function virtualization.

CHEN SUN (c-sun14@mails.tsinghua.edu.cn) is a Ph.D. student at the Institute for Network Sciences and Cyberspace, Tsinghua University, China. His research fields include software-defined networking and network function virtualization.

GUANG YAO (yaoguang@cernet.edu.cn) received B.S. and Ph.D. degrees in computer science from Tsinghua University, China. He is a postdoctoral researcher at the Institute for Network Sciences and Cyberspace Tsinghua University. His research fields include Internet architecture, software-defined networking, and source address validation.

HONGXIN HU (hongxih@clemson.edu) is an assistant professor in the Division of Computer Science, School of Computing, Clemson University. His current research interests include security in SDN and NFV, security and privacy in social networks, and security in cloud and mobile computing. He received his Ph.D. degree in computer science from Arizona State University in 2012.