Hindawi
Security and Communication Networks
Volume 2018, Article ID 5650205, 13 pages
https://doi.org/10.1155/2018/5650205

Research Article

FAS: Using FPGA to Accelerate and Secure SDN Software Switches

Wenwen Fu, Tao Li, and Zhigang Sun

College of Computer, National University of Defense Technology, Changsha, Hunan 410073, China

Correspondence should be addressed to Wenwen Fu; fuwenwen94@163.com

Received 12 October 2017; Accepted 17 December 2017; Published 17 January 2018

Academic Editor: Chengchen Hu

Software-Defined Networking (SDN) promises the vision of more flexible and manageable networks but requires certain level of programmability in the data plane to accommodate different forwarding abstractions. SDN software switches running on commodity multicore platforms are programmable and are with low deployment cost. However, the performance of SDN software switches is not satisfactory due to the complex forwarding operations on packets. Moreover, this may hinder the performance of real-time security on software switch. In this paper, we analyze the forwarding procedure and identify the performance bottleneck of SDN software switches. An FPGA-based mechanism for accelerating and securing SDN switches, named FAS (FPGA-Accelerated SDN software switch), is proposed to take advantage of the reconfigurability and high-performance advantages of FPGA. FAS improves the performance as well as the capacity against malicious traffic attacks of SDN software switches by offloading some functional modules. We validate FAS on an FPGA-based network processing platform. Experiment results demonstrate that the forwarding rate of FAS can be 44% higher than the original SDN software switch. In addition, FAS provides new opportunity to enhance the security of SDN software switches by allowing the deployment of bump-in-the-wire security modules (such as packet detectors and filters) in FPGA.

1. Introduction

Software-Defined Networking (SDN) is a transforming networking design that simplifies network management and improves programmability of network [1]. Its basic attributes include the separation of control and data planes, logically centralized control of networks, and flexible and open interface to program underlying network infrastructure. In the architecture of SDN, the southbound interface is responsible for the interaction of network states between the control and data planes. Furthermore, it defines the forwarding abstraction of SDN data plane.

As SDN offers some hope for rapid prototyping and deployment, the data plane must provide mechanisms to deploy new network protocols, header formats, and functions, yet it still forwards traffic as fast as possible. OpenFlow is the most widely used standard SDN southbound interface which is a vendor-independent interface to switching elements [2, 3]. Most SDN switches are thus OpenFlow SDN Switches.

There are mainly two ways to implement the SDN data plane: hardware and software. Some SDN hardware switches, developed by the vendors of HP, NEC, and Arista, utilize customized ASIC switching chip. Although the ASIC way can get high forwarding performance, it can hardly be extended for new protocols and functions. Programmable hardware switches [4, 5] can provide the flexibility at the expense of large amount of hardware resources, such as expensive TCAMs for flow entries.

The SDN software switches are the most flexible to support new network services, new protocols, and new functions [6–9]. Unlike the TCAMs in hardware SDN switches, memory resource for accommodating flow rules is abundant in software SDN switches [10]. Thus, they are primary choices for SDN researchers in laboratories and have been widely deployed as first-hop virtual switches in data centers. However, a purely software-based approach can hardly satisfy the strict performance and line-speed security requirements of most modern networks.

With the continuous improvement of the capacity and the computing power of FPGA (Field Programmable Gate Array), we try to exploit benefits of FPGA on processing packets. It is noteworthy that Microsoft has built FPGA fabric attached to each server to accelerate large-scale data center services with customized function logic [11]. Due to its high performance with flexible reconfigurability, FPGA is also a good choice for accelerating and securing SDN software switches. We propose FAS (FPGA-Accelerated SDN software switch) to enable the offloading of time-consuming software functional modules and implementation of the real-time security modules in SDN switch processing path. The mechanism can address the performance shortage nicely while retaining the flexibility of software switches. In addition, it can also enhance the security property of software switches by deploying bump-in-the-wire security modules in FPGA. The contributions of this paper are summarized as follows.

(1) We make a comprehensive survey on current SDN switches in both academia and industry and classify the existing implementation models of SDN switches into different categories.

(2) We analyze the bottlenecks in SDN software switches on modern commodity multicore platform.

(3) We design FAS mechanism to offload functions in the forwarding path of SDN software switch, including packet buffer management, packet parsing, and some action executions for packets, to FPGA hardware.

(4) We implement the prototype of FAS on NetMagic-Pro, an FPGA-based network processing platform, and compare the performance with the original SDN software switches on commodity multicore platform.

The remainder of the paper is organized as follows. In Section 2, we review the evolvement of OpenFlow specification and introduce current SDN switches and their implementation models. Section 3 analyzes the overhead of OpenFlow forwarding in SDN software switches and points out the bottlenecks. In Section 4, we put forward a mechanism to offload software procedures of OpenFlow forwarding path to FPGA. Section 5 describes detailed design of the FAS mechanism implemented on an FPGA-based network processing platform (i.e., NetMagic-Pro). In Section 6, we give the performance comparison of FAS and the original SDN software switch. We summarize our work in Section 7.

2. Background and Related Work

In this section, we firstly make a brief introduction of the evolution of OpenFlow protocol, which challenges the design of SDN switches. Then we describe implementation models of SDN switches in both academia and industry.

2.1. Evolvement of OpenFlow. OpenFlow is the first and prevailing standard defined by Open Network Foundation (ONF) among all SDN southbound interfaces. It is wildly supported by many commercial switches, including HP, NEC, Arista, and Pica 8, and the list is still continually growing.

Since the first version (v1.0) distributed by ONF in December 2009, the specification of OpenFlow has been updated to version 1.5 in 2015 and has grown increasingly more complicated [3]. Although many features have been added to OpenFlow, the core concept of OpenFlow has not changed. OpenFlow switches process and forwards traffic on the basis of flows instead of individual packets. In the first version, the data plane abstraction is a single flow table of flow rules which could match packets on 12 header fields (e.g., MAC addresses, IP addresses, and TCP/UDP port numbers). In version 1.5, the OpenFlow switch has a pipeline of flow tables, where each flow rule has match fields (41 fields in the packet header), instructions (e.g., drop, flood, forward, or send the packet to the controller), a set of counters (to track the number of bytes and packets), a priority (to disambiguate between rules with overlapping patterns), timeouts (expiration time of flow rules), cookie (opaque data value chosen by the controller), and flags (to alter the way flow entries are managed). Upon receiving a packet, an OpenFlow switch identifies the highest-priority matching rule, performs the associated actions, and increments the counters.

With the evolvement of OpenFlow specification, OpenFlow has been extended with more capabilities and functions. As a result, the procedure of OpenFlow processing is getting more complicated and proliferating with no sign of stopping. It makes challenges to the implementation of SDN switches.

2.2. Implementation Models of SDN Switches. According to the OpenFlow specification, an SDN switch usually consists of four components: OpenFlow Channel (OFCh), OpenFlow Forwarding pipeline (OFFw), and physical port (Port). Besides, to accelerate multiple-tuple classification procedure in OFFw, a commonly used technique, Flow Cache (FCa), is introduced in SDN switches [12]. The descriptions of the four modules are as follows.

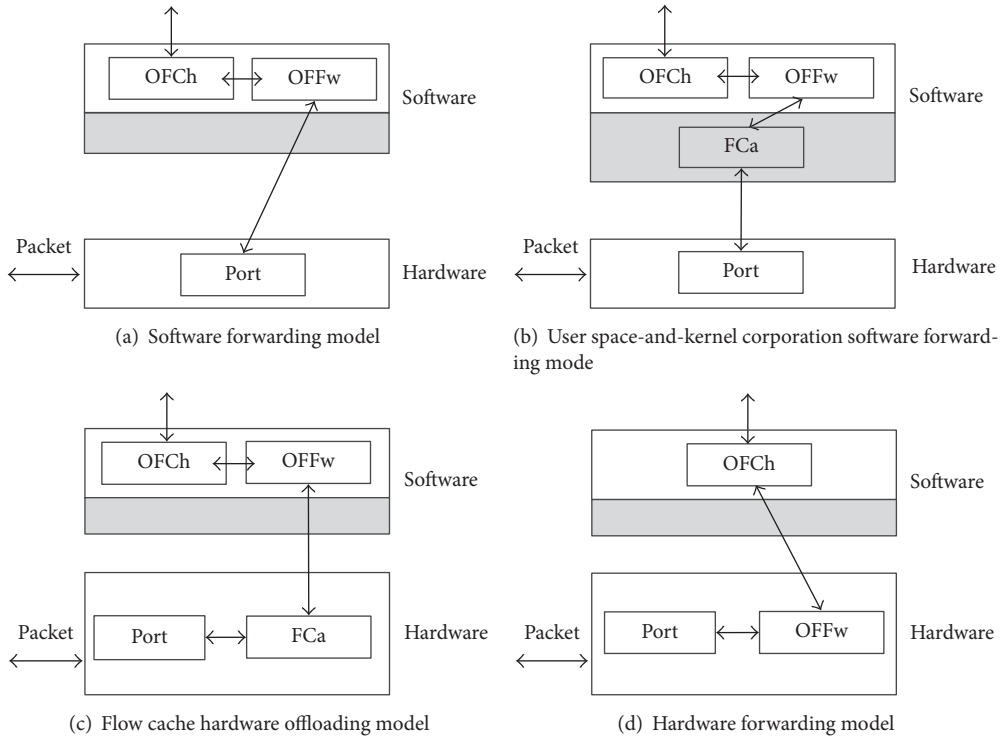
Port. It is the interface between the network and the switch and is responsible for packet receiving and sending.

FCa. It is the fast forwarding path for OFFw, which caches the entries recently matched in OFFw. Since network traffic has sufficient locality to provide high cache hit rate with relatively small cache size, FCa can accelerate the forwarding rate by bypassing OFFw.

OFFw. It maintains flow table of entries, in which each entry contains a set of packet fields to match and the corresponding actions to perform (e.g., forwarding, dropping, and modifying header).

OFCh. In the event when a switch does not find a match in OFFw, the packet is forwarded to the controller through OFCh. After deciding how to forward the new flow, the controller sends OpenFlow messages to the required switches. Then OFCh resolves those messages, generates flow rules, and installs them to the flow table. Besides, OFCh is also responsible for exchanging the states between the controller and the switch.

SDN switches have been implemented on different platforms (e.g., general CPUs, fixed function switch ASICs, reconfigurable hardware, NPUs, and FPGAs). The platforms can be divided into two categories: hardware-based switches



(a) Software forwarding model

(b) User space-and-kernel corporation software forwarding mode

(c) Flow cache hardware offloading model

(d) Hardware forwarding model

Figure 1: Current implementation models of SDN switches.

and software-based switches. Figure 1 shows different implementation models of the existing SDN switches. In the hardware-based switches, FCa or OFFw is implemented by hardware, such as ASICs and FPGAs. For example, Naous et al. implement an OpenFlow switch on Stanford's NetFPGA platform [13]. Pongrácz et al. devise an NP-based SDN switch to enhance the programmability in the data plane [14]. Some companies, such as Pica 8, supply commodity OpenFlow switches based on ASICs switch chips. These hardware switches are based on the Flow Cache hardware offloading model (in Figure 1(c)). RMT [4] and FM6000 [5] devise reconfigurable match tables in the OpenFlow pipeline, and they conform to the hardware forwarding model in Figure 1(d). The SDN hardware switches can provide sufficient forwarding capability, but they are costly and inflexible.

Recent improvement in processing power of multicores has given reason to revisit software switching. Due to its high flexibility and short development cycle, the SDN software switches are getting increased attention. Software switches commonly use commodity off-the-shelf (COST) PCs or servers equipped with multicores and multiple Network Interface Cards (NICs), running on general-purpose operation systems (such as Linux). The general-purpose OS provides a comfortable environment for research and development. OpenFlow reference switch (maintained by ONF) [6], OFSoftSwitch (maintained by CPqD) [7], and OpenFlow Click (maintained by Stanford) [8] are the typical SDN software switches referred to the software forwarding model (in Figure 1(a)). Open vSwitch [9] (namely, OVS, maintained by VMware) is an SDN software switch belonging to the user space-kernel cooperation model as shown in Figure 1(b). Table 1 gives the detailed information of above-mentioned SDN switching platforms.

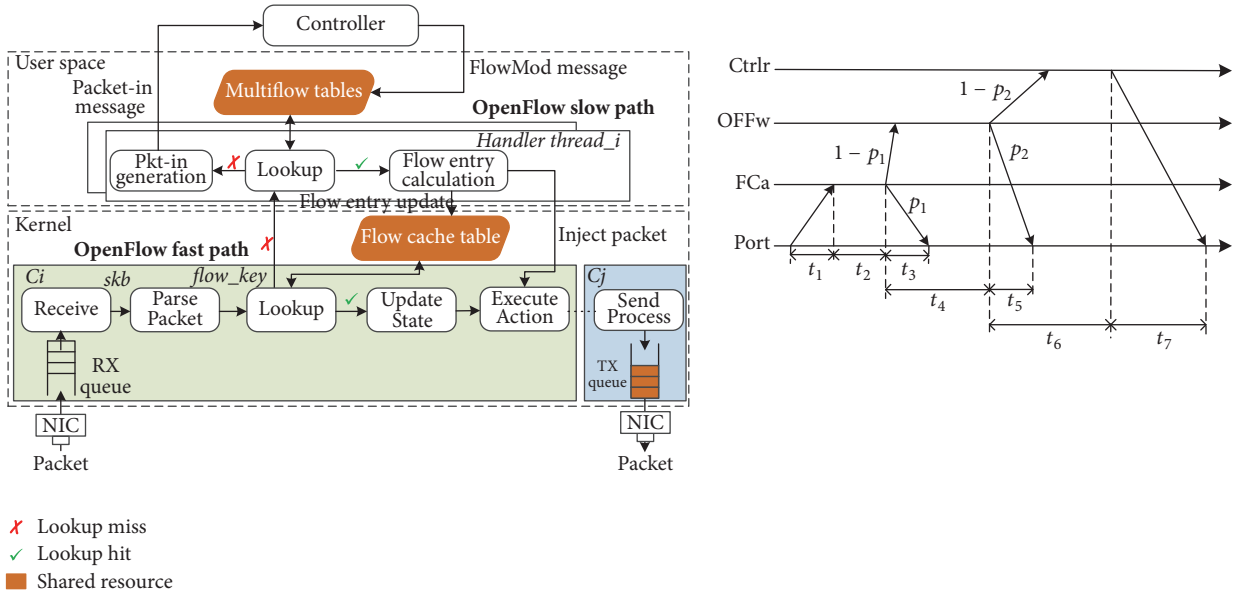
3. Problem Description and Analysis

3.1. OpenFlow Forwarding in SDN Software Switches. As the OVS is the most advanced and widely used SDN software switch, we focus on the user space-and-kernel cooperation (UKC) model as shown in Figure 1(b). Figure 2(a) illustrates the forwarding process of UKC model, while Figure 2(b) shows the timeline of it. For simplicity, we first discuss the procedure under multicore architecture with single-queue Network Interface Cards (NICs). The case with multiple-queue NICs will be discussed in Section 3.2. Without loss of generality, we assume that NET_RX and NET_TX of hardware interrupts in NIC i are both served by a fixed CPU core i. Cores i and j are denoted as Ci and Cj in Figure 2(a). If i equals j, packets are sent and received through the same NIC and processed by the same CPU core.

When a packet arrives at NIC i, this packet is attached to a descriptor in the NIC i's receiving (namely, RX) queue. The descriptor indicates the memory locations to store the incoming packets via Direct Memory Access (DMA) transfer. There are two data structures in the network stack of Linux Kernel. data_buff of 2 KB size is to hold the packet itself. The other data structure is sk_buff, which carries packet information metadata (e.g., pointer of packet data, MAC header, IP header, and packet states) used by TCP/IP protocol stack. The size of sk_buff is about 250 bytes. The sk_buff has a

Table 1: SDN switching platforms.

Names	Supporting OF version	Languages/hardware type	Application scenarios	Reference model
NetFPGA-based OpenFlow switch	1.0	FPGA	Research	Figure 1(c)
NP-based OpenFlow switch	1.0	Network processor	Research, enterprise	Figure 1(c)
Commodity OpenFlow switch	1.0	ASIC	Research, enterprise, data center	Figure 1(c)
RMT	All	ASIC	Research, enterprise	Figure 1(d)
FM6000	All	ASIC	Research, enterprise	Figure 1(d)
OpenFlow reference switch	All	C	Research	Figure 1(a)
OFSoftSwitch	After 1.3	C	Research	Figure 1(a)
OpenFlow Click	All	C++	Research	Figure 1(a)
Open vSwitch	All	C & kernel C	Research, virtualized data center	Figure 1(b)



(a) Forwarding process of UKC model

(b) Timeline of UKC forwarding

Figure 2: UKC implementation models of SDN software switch.

pointer to data_buff, and it makes up a Skb with the data_buff. After hardware interrupt of packet incoming is served, the Software Interrupts (SoftIRQs) are scheduled afterwards to accomplish the subsequent packet processing. The SoftIRQs are also bound to the specific processors.

In OVS, the handler function of SoftIRQ of *Ci* parses the packet by extracting all related match fields from Skb and stores them to flow_key.

The flow_key is used to look up the Flow Cache, which is shared among all processors in the kernel. If the Flow Cache contains the flow_key value, the Flow Cache will return instructions for processing the packet. Then it updates corresponding states and executes actions to the packet. If the packet is to be delivered by NIC j, it will be placed in the sending (namely, TX) queue of the NIC j. The sending process will be called in the function of NET_TX SoftIRQ of *Cj*.

If the lookup of Flow Cache is missed, the packet will be sent to the user space of Linux via the communication mechanism between the kernel and the user space, for example, netlink. Multiple handler threads in the user space will be created when the OpenFlow software switch is set up. These threads are responsible for processing mismatched packets coming from the kernel. The handler threads look up the shared multiflow tables. If the packets match with the tables, the handler threads will calculate for new Flow Cache entries and update them into the Flow Cache. The handler will also reinject the packet back to kernel for executing corresponding actions on the packet. If not, a Packet-in message will be generated and sent to the controller. The controller responds

with a FlowMod message to install corresponding flow rule for the packet to the multiflow tables.

Based on the analysis of the forwarding process, we evaluate the processing overhead in SDN software switches. We make some notations to define the time slots in the procedure. The time period from the packet arriving at the NIC to the lookup of the cache is denoted as t_1. The time of Flow Cache lookup is t_2. The time period from the start of matching Flow Cache to the end of sending out the packets is t_3. In the lookup, we denote the cache hit rate as p_1. The time period from the start to the end of looking up multiflow tables of user space is t_4. The time period from the beginning of matching multiflow tables to the end of sending out the packet is t_5. We denote the hit rate of multiflow tables as p_2. The time period from the start of triggering the controller to the end of installing the rule is t_6. At last, the time used for sending out the packet is t_7. Given these notations above, the evaluated total processing time T is formulated as follows:

$$T = t_1 + t_2 + p_1 t_3 + (1 - p_1)\, p_2\, (t_4 + t_5) + (1 - p_1)(1 - p_2)(t_6 + t_7). \quad (1)$$

Due to the high locality of network traffic and the proactive flow rule setup, the Flow Cache can get a high hit rate. As verified by Pfaff et al., the overall cache hit rate of Open vSwitch was 97.7% (i.e., the value of p_1) in a real commercial multitenant data center [15]. Thus, in most instances, the total process time T of a packet in SDN software switches can be approximately calculated as

$$T = t_1 + t_2 + t_3. \quad (2)$$

These three periods constitute the fast path of the OVS.

As depicted in Figure 2(a), the OpenFlow fast path consists of six functional modules: Receive, Parse Packet, Lookup, Update State, Execute Action, and Send Process. To understand the overhead for each functional module, we run Open vSwitch in a commodity PC as a software switch. We use the tool of Iperf [16] for generating the input traffic which is running over two 1 GbE NICs of the PC. Then we use Oprofile [17] to count CPU cycles for each function in fast path of Open vSwitch. We group all functions into the above six functional modules and the experimental results are shown in Figure 3. As we can observe, the Lookup module is the major bottleneck in the fast path. The other modules consume up to 44% of processing time in total.

3.2. Bottlenecks in SDN Software Switch. There are three main bottlenecks (i.e., packet I/O, software forwarding, and OpenFlow classification) in SDN software switch.

Packet I/O. Unlike CPU-intensive tasks, packet I/O is a critical step in software packet forwarding. Packet receiving and sending consume a large amount of CPU cycles. As pointed out in [18], the buffer allocation and release in packet I/O are the two major overheads, which represent up to 54% of total overhead.

Researchers have proposed several optimizing techniques such as Skb recycle queue, memory mapping, batch processing, affinity, and software prefetching to accelerate packet

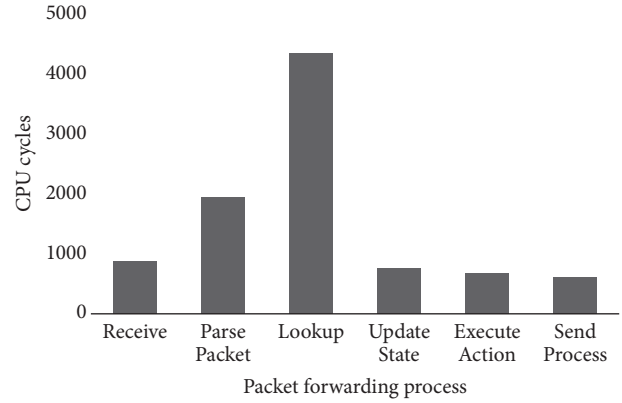


Figure 3: Packet forwarding process overhead breakdown. The processing for packets in Execute Action module only contains the forwarding operation.

I/O. By combining some of above techniques, Intel develops a high-performance packet processing architecture on x86 platforms, named Data Plane Development Kit (DPDK) [19]. But the DPDK framework needs the support of specific CPU and NIC, which is not general. Rizzo designs a novel framework for fast packet I/O in general-purpose OS, named netmap [20]. However, netmap is originally run in the user space of OS. The performance drops when it is applied to kernel-based SDN software switches.

Software Forwarding. A critical problem is contention for shared resources—caches, queues in NICs, and flow tables—in the case of multiple threads running concurrently to forward packets [21]. In order to improve the processing performance of SMP Linux, NIC-based core affinity is proposed to exploit the parallel packet processing capability of multicore architecture. By maintaining the affinity relation of a core and an NIC, software and hardware interrupts of the NIC are handled by the specified core. As a result, it incurs less cache miss and improves the execution efficiency of packet processing. However, as for packet forwarding, the NIC-based core affinity is not efficient. Since the receiving and sending of packet are handled in different cores usually, which causes the problems of mutex exclusion and cache coherence, Han et al. propose queue-based core affinity [22]. Each receiving and sending queue in an NIC maps to a core, and the corresponding CPU core accesses the queue exclusively, eliminating cache bouncing and lock contention caused by shared data structures. However, the NICs must support multiple queues and Receive Side Scaling (RSS), which can hardly be scalable.

OpenFlow Classification. As described in Figure 2(a), the OpenFlow classification in the kernel consists of the operations of packet parsing, Flow Cache lookup, counter updating, and action execution. Packet classification is time-consuming on general-purpose processors, and packet classification becomes even worse. According to the specification of OpenFlow 1.5, 41 fields of packet should be parsed, extracted, and looked up during the packet classification. That makes OpenFlow classification extremely complex and be of

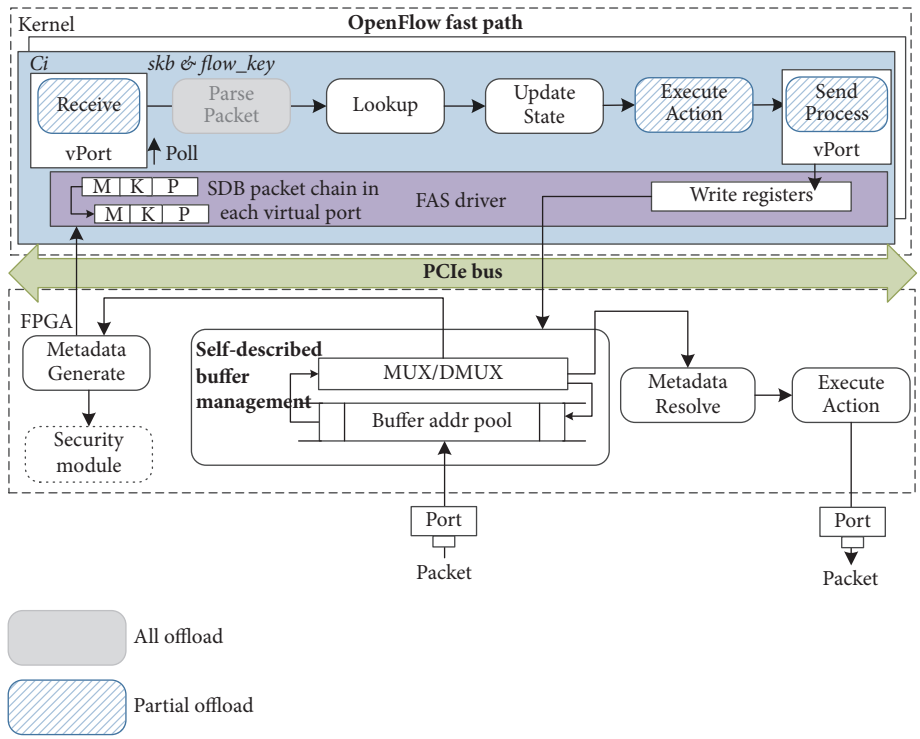


Figure 4: The framework of FAS mechanism.

low performance. Thus Putnam et al. designed three-layer cache architecture (microflow cache, megaflow cache, and OpenFlow pipeline) for accelerating OpenFlow classification [11]. Moreover, some actions of packet processing are costly in software, such as fields rewriting and packet encapsulation/decapsulation.

In addition, the performance of the SDN software switches will be largely decreased if complex security functions are required to be integrated. Most processing functions involved in security functions are stateful and CPU-consuming. The method to improve the security capacity without hurting performance should be investigated for SDN software switch.

4. FAS Mechanism

4.1. Architecture of FAS. Aiming at the above-mentioned bottlenecks of existing SDN software switches, we exploit programmable hardware, FPGA, to accelerate the rate of computation. FAS mechanism is designed to offload time-consuming functional modules in the OpenFlow software fast path to FPGA.

As depicted in Figure 4, the FAS mechanism consists of three components in FPGA. Self-Described Buffer Management module is used to offload some time-consuming parts in packet receiving and sending of the Linux kernel. Metadata Generate module is to offload the procedure of parsing packet. Execute Action is used to perform some actions after acquiring packet and its actions from Metadata Resolve module. Metadata Resolve module receives packet processing metadata information from software, resolves it, and notifies Execute Action module.

Packet Buffer Management Offloading. The management of Skb for each packet is a critical operation during packet I/O. It contains the operations of conversion from the raw packet to Skb, initialization of Skb, and allocation and deallocation of Skb. It consumes majority of CPU cycles in the procedure of packet I/O. Previously, our research group has proposed Self-Described Buffer (SDB) management in hardware to eliminate the packet buffer management expenditure in software [23]. In SDB, the original separated data structure Skb—packet and its metadata—is merged into a successive stored packet buffer (we call it the SDB packet buffer). The software preallocates fixed size space for SDB packet buffers in main memory at initial phase. This enables the SDB hardware to be able to dynamically allocate and recycle circular addresses of the SDB packet buffers during packet I/O. The packet buffer management overhead of software is thus eliminated. FAS makes use of SDB to alleviate the bottleneck of packet I/O.

Packet Parsing Offload. The OpenFlow fast path extracts matching fields to generate *flow_key* by parsing *Skb*. And the parsing procedure is the second most costly part as illustrated in Figure 3. We thus offload the operation of packet parsing to hardware and put the parsing result in the metadata of the packet. The parsing process is relatively straightforward in OpenFlow forwarding, as depicted in Figure 5. It can be easily implemented in FPGA hardware.

Action Execution Offload. The Execute Action module in FPGA executes partial actions for packet according to hardware capability. The fundamental operation is the forwarding action to corresponding port. In the sending process, the

Table 2: Comparisons of acceleration mechanisms for SDN software switches.

Names	Method	Acceleration object	Complexity	Network virtualization	HW state manage. cost
DPDK-based OF switching	Intel CPU/NIC	Packet I/O	Low	Support	No
Netmap-based OF switching	Netmap lib	Packet I/O	Low	Support	No
Flow Director	Driver modification	OF flow cache	Medium	Not support	High
SSDP	Commodity switching chip with TCAM	OF forwarding path	High	Not support	High
FAS	FPGA	Time-consuming functional modules in OF fast path	Medium	Support	Low

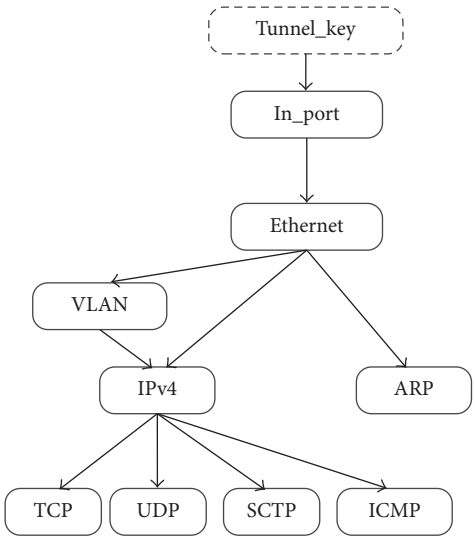


Figure 5: A parse graph example for OpenFlow.

hardware will resolve the packet address written in the FPGA registers and read the packet with metadata by DMA. The parameters of actions are carried in metadata, which are specified by the software. For example, if the queue action in metadata is resolved by the module of Metadata Resolve, the packet will be enqueued to the specified queue of corresponding port in FPGA. The offload of Execution Action can eliminate complex and time-consuming software packet operations, such as packet field rewriting and encapsulation/decapsulation.

Security Function Extension Interface. To support hardware security functions extension, we have proposed a well-defined module interface. The downstream and upstream interfaces of security modules are FIFO-liked interfaces and the control command is encoded into the metadata in front of the packet data. The security module can be easily embedded into the FPGA processing pipeline if the module interface definition is confirmed.

4.2. Comparisons between FAS and Existing Mechanisms. Many mechanisms have been proposed to improve the performance of OpenFlow software switching. We list state-of-the-art mechanisms and show their differences with FAS in Table 2. DPDK and netmap provide high-performance I/O framework for OpenFlow software switching. Note that DPDK relies on specific CPUs and NICs (e.g., Intel 82599), while netmap can be used for general NICs [24]. Flow Director proposes using the packet classification hardware on the NIC as OpenFlow fast forwarding path [25]. But the cost of maintaining the Flow Cache entries in NIC driver is relatively high. SSDP proposes enhancing slow software forwarding path with commodity switching chip including TCAM [26]. The common OpenFlow forwarding path is divided into two data planes in SSDP: macroflows in switch chip and microflows in CPU. Maintaining state consistency between the two data planes is intractable.

5. Preliminary Implementation

This section describes a design and preliminary implementation of FAS on NetMagic-Pro (NMP), which is an FPGA-based network processing platform with multicore CPU.

5.1. Brief Introduction of NetMagic-Pro. The hardware and software in NMP (as shown in Figure 6) are both programmable for packet processing. The total power consumption of NMP is 85 W. NMP consists of four parts: CPU board, FPGA board, line-card board, and power supply. The CPU board integrates an Intel i7-4700EQ CPU with 4 GB memory. The FPGA board is equipped with Altera EP4SGX180 and a piece of Flash storing the configuration file of the FPGA. The software running on multicore CPU communicates with FPGA through the PCIe bus. The PCIe v2.0 with 8 lanes offers a high link bandwidth of 40 Gbps in each direction. The line-card board provides eight 1-GigE Ethernet ports.

Similar to NetFPGA, NMP enables researchers and students to experiment with Gigabit rate networking hardware. The difference is that NMP provides better programmability in both hardware and software and higher software-hardware communication performance than NetFPGA. We implemented FAS mechanism on the platform of NMP.

5.2. FAS Driver. A kernel driver in the software is designed for FAS. It is responsible for NMP packets (including metadata, flow key, and packet data) communicating between the

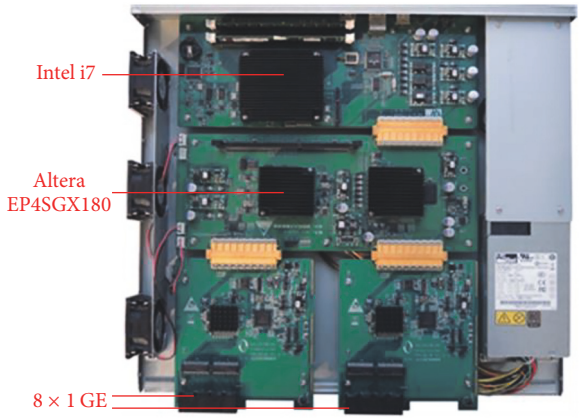


Figure 6: NetMagic-Pro network platform.

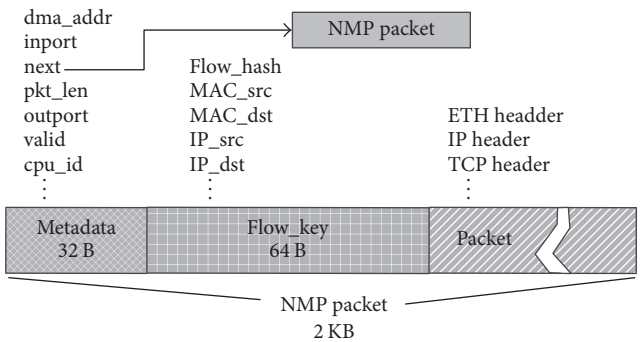


Figure 7: The format of NMP packet.

software and hardware. The data structure of NMP packet is illustrated in Figure 7. Instead of packet descriptors, all control messages for the packet are stored in metadata, including DMA address, ingress port, next NMP packet address, and packet valid indicator. *Flow_key* of 64 bytes consists of all matching fields extracted by hardware, whose size is 64 B. *Packet* accommodates the packet head and body, which is usually less than 1518 bytes. Therefore, 2 KB memory is preallocated for each NMP packet by software. FAS driver utilizes two techniques to optimize the performance of packet I/O and reducing cache miss rate in packet forwarding.

Polling Instead of Interrupt. Different from the hybrid of interrupt and polling in Linux NAPI (New API), we adopt polling approach to fetch the incoming NMP packets in FAS driver. It is reasonable since NMP is a packet forwarding platform. The FAS driver polls NMP packet from hardware DMA. The packet valid flag in the metadata is used to indicate whether the packet is ready to be processed. The NMP packets are organized into chain to simplify the polling. The next NMP packet address points to the next packet to be processed.

Core Affinity in Packet Dispatch. Each polling thread only runs on one CPU core. NMP packets are organized in chains; each chain is only assigned to one thread, namely, one core, for processing. With Run-to-Completion (RTC) mode of thread, each packet is received and transmitted by one core. It thus reduces the context switching overhead among different cores during processing. And it also reduces TLB update in accessing packet buffer.

Table 3: Basic functions for the virtual Ethernet port.

Interface of function	Description
static int nmp_open(struct net_device *netdev)	Open a port
static int nmp_close(struct net_device *netdev)	Close a port
static int nmp_xmit_frame(struct sk_buff *skb,struct net_device *netdev)	Send packet to a port
static struct net_device_stats *nmp_get_stats(struct net_device *netdev)	Acquire port statistics
static int nmp_set_mac(struct net_device *netdev,void *p)	Set MAC address of a port
static int nmp_change_mtu(struct net_device *netdev,int new_mtu)	Set MTU of a port

5.3. Virtual Ethernet Port. Virtual Ethernet ports are required for OpenFlow software switches to fully utilize the features of FAS. In NMP, the Ethernet ports are not standard commodity NICs. Therefore, we implement a customized network device for virtual Ethernet ports in NMP, which is applied to exchange packets and states for OpenFlow software switches.

The virtual Ethernet port provides basic functions for packet forwarding applications, including network device application, packets sending/receiving, port counting, MAC address configuration, and Maximum Transmission Unit (MTU) configuration. Thus, the virtual Ethernet ports of NMP transparently support all features of OpenFlow software switches running on NMP. The core operation of customized network device is the conversion between *Skb* data structure and NMP packet. As shown in Section 6.2, the cost of conversion is quite low.

There are three steps to implement the virtual Ethernet port for NMP. In the first step, we allocate a new network device by calling *alloc_etherdev* in the kernel. In the second step, we implement basic functions for the standard Ethernet port. The functions are listed in Table 3. All these functions are defined in the data structure of *net_device_ops*.

Among these functions, the most important one is packet sending, as shown in Pseudocode 1. There are two types of packets to be handled. For the packet received by FAS driver, if it is forwarded to some port; it will be converted from *Skb* to NMP packet. For the packet generated by the software, for example, OpenFlow messages sent to the controller, it is not provided with preallocated hardware-maintained addresses and other related information. It requires a new generated NMP packet with "soft" tag marked. Then the two types of the packets can be sent by writing hardware registers with corresponding metadata. The FPGA hardware executes operations on the packets according to their metadata.

In the third step, we register the new network device to the kernel. After configuring related parameters for the virtual

```
Pseudocode: Sending procedure in NetMagic-Pro with FAST
(1) nmp_xmit_Frame (Skb, nmp_netdev){
(2)    if (Skb->flag == NMP_PACKET){
(3)        nmp = transfer (Skb, nmp_netdev);
(4)        send_nmp_pkt (nmp){
(5)            //hardware send function by writing registers
(6)            NMP_SEND_PKT_REG(REG_addr,nmp.metedata);
(7)        }
(8)    }
(9)    else{
(10)       nmp = get_soft_nmp(Skb, nmp_netdev);
(11)       nmp.metedata.soft = 1;
(12)       send_nmp_pkt (nmp){
(13)           NMP_SEND_PKT_REG(REG_addr,nmp.metedata);
(14)       }
(15)   }
```

Pseudocode 1: Sending pseudocode for virtual Ethernet port.

Table 4: Components of platforms in experiment.

	NetMagic-Pro	Commodity PC
CPU	Intel i7-4700EQ 2.4 G	Intel i7-3770 3.4 G
The number of cores	4	4
Cache	6 MB L3-cache	6 MB L3-cache
DRAM	4 GB DDR3 SDRAM	4 GB DDR3 SDRAM
NIC	1 Gbps x8 NMP port	1 Gbps x2 Intel 82579
OS	Ubuntu 14.04 LTS	Ubuntu 14.04 LTS
FPGA	Altera EP4SGX180	—

Ethernet port, the *register_netdev* function is called in the kernel to complete the registration of the network device. Then the virtual Ethernet port can be accessed in the user space via the system command "ifconfig." At this time, we can create OpenFlow software switches on NMP. For example, the command "ovs-vsctl add-port br0 nmp1" is used to add port 1 in NMP to Open vSwitch's bridge.

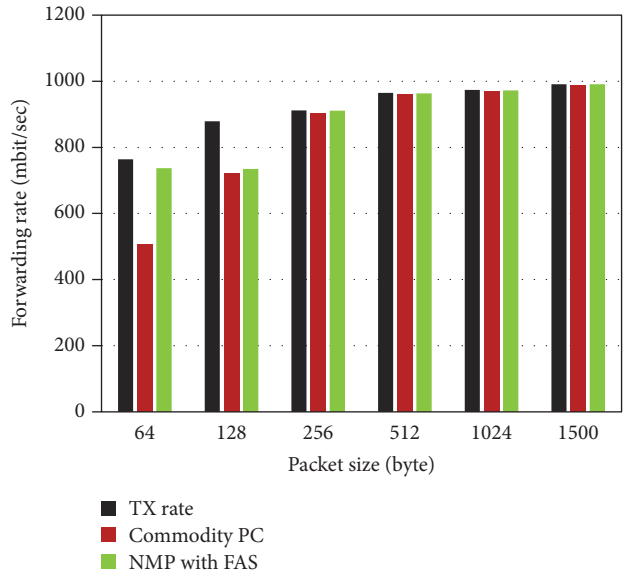


Figure 8: The comparison of forwarding rate with one rule.

6. Experiment Evaluation

6.1. Experiment Setup. We use Open vSwitch (version 2.3.1) [9] as the typical SDN software switch in experiments. Open vSwitch runs on NMP with FAS enabled and a commodity PC. Both platforms are directly connected to IXIA XM2 Tester with two ports/NICs. The two ports in IXIA XM2 Tester are used for traffic source and sink, respectively. The packet size of testing flows can be configured in IXIA XM2 Tester.

Table 4 lists the specification of the experimental components. Both devices under test have almost the same hardware configuration except for the CPU. Note that the performance of the PC's CPU (Intel i7-3770) is a little better than that of NMP (Intel i7-4700EQ). In the experiments, we mainly evaluate the efficiency of FAS mechanism from two aspects: forwarding rate and forwarding latency. We believe that PCIe bandwidth for transmitting packets between the software and FPGA is not a bottleneck, because PCIe 2.0 x8 links provide 40 Gbps, which is far more enough than required for unidirectional traffic generated in the experiment.

6.2. Forwarding Performance Evaluation. Forwarding performance of OpenFlow software switching is concerned with packet size and flow rules. Firstly, we test the forwarding rate of Open vSwitch under 1 flow rule. All tests last 60 seconds and forwarding rates are sampled by 1 second. The average forwarding rates are shown in Figure 8. When forwarding traffic is with the size of 64 B Ethernet packets, NMP with FAS achieves throughput of 740 Mbps, about 97% of the theoretical wire-speed (762 Mbps). NMP with FAS gets 44% higher performance than the commodity PC. When the packet size is 128 B, the forwarding rate of NMP with FAS achieves wire-speed forwarding, which is 18.5% higher than the commodity PC. The gap of performance between FAS and

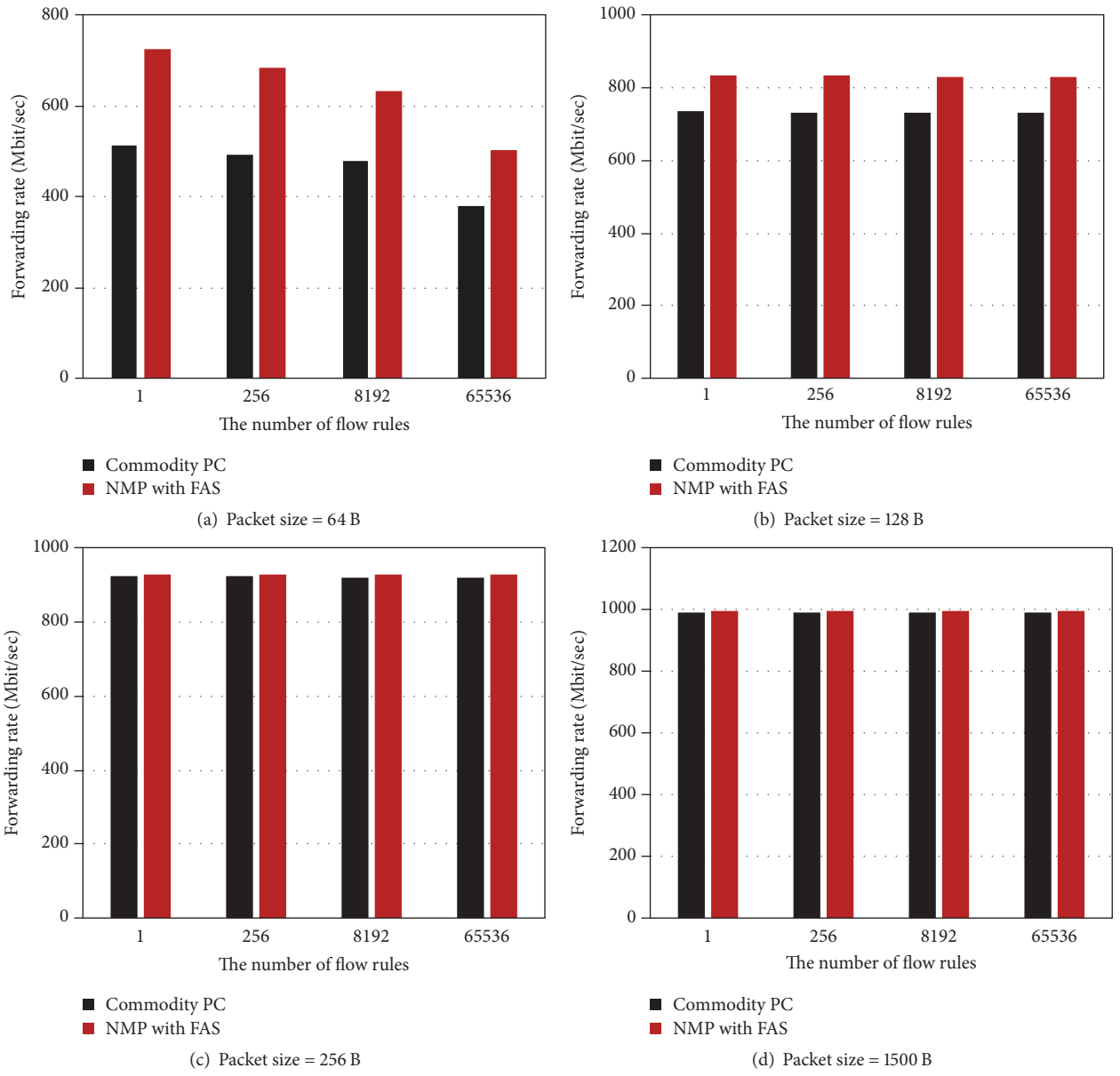
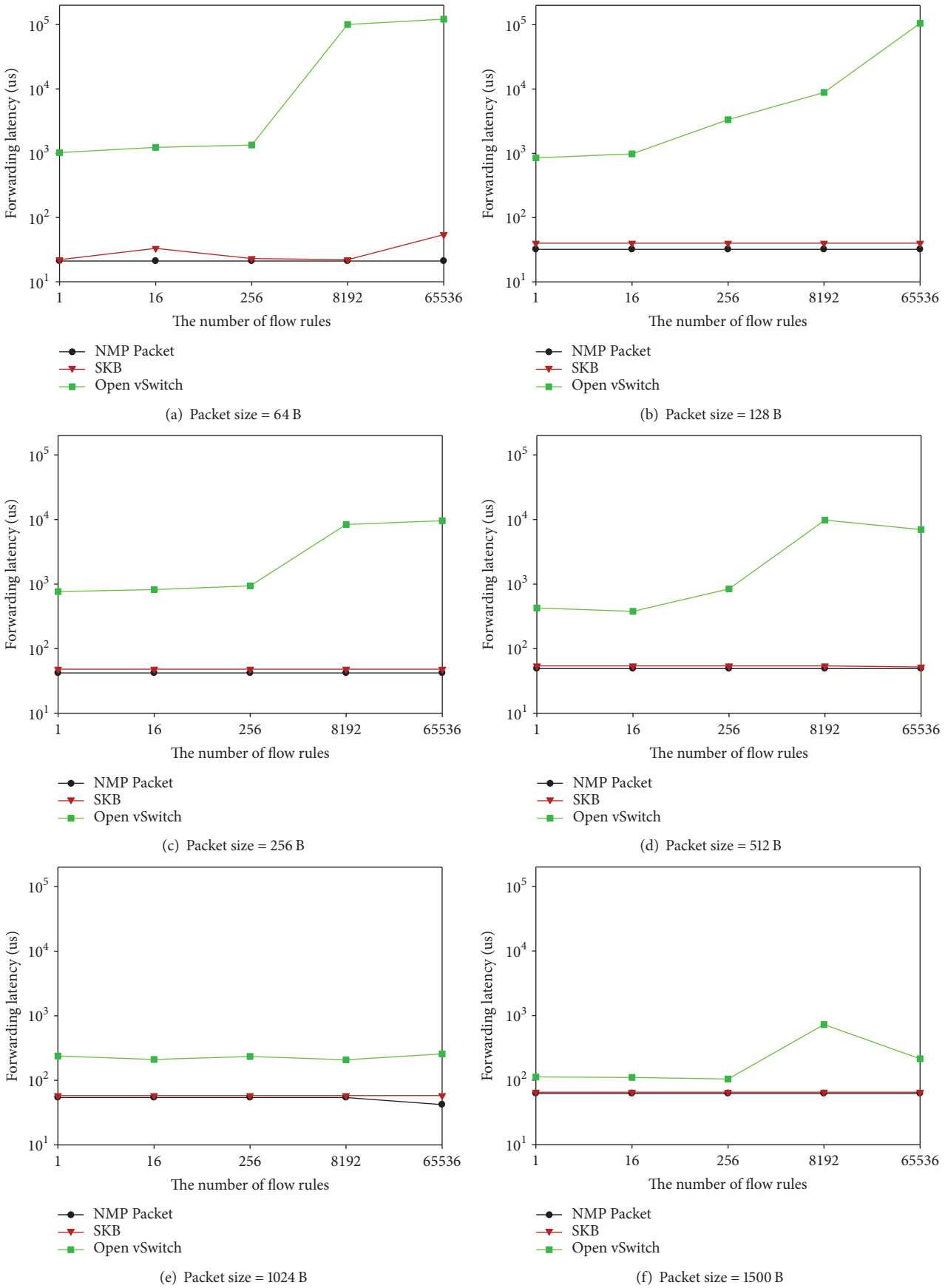


Figure 9: The forwarding rate with various number.

the original OVS becomes smaller when the size of packets increases. That is because both can achieve wire-speed for large packets. We can conclude that, in the case of 1 flow rule, NMP with FAS can get nearly wire-speed forwarding rate even for minimum size packets, which significantly outperforms the commodity PC.

Secondly, we make comparisons of the forwarding rate under different number of flow rules. The preinstalled flow rules are all mutually exclusive with others. The numbers of rules are 256, 8192, and 65536 in different experiments. The traffic is generated according to the rules to guarantee that every packet matches the corresponding flow rule. We also vary the size of packets. The experimental results are shown in Figure 9. For the cases of 512 B, 1024 B, and 1500 B Ethernet packet, the forwarding rates of NMP with FAS and the commodity PC have little difference and both approach the wire-speed under all different numbers of flow rules. The data of 512 B and 1024 B are omitted for simplicity. For the packet size of 64 B (Figure 9(a)), the forwarding rate goes down with increasing flow rules. When the number of flow rules reaches 65536, the forwarding rates of NMP with FAS and the commodity PC fall by 45% and 47%, respectively, when compared to one-rule case, but the forwarding rate of the NMP with FAS is still 44% higher than that of the commodity PC. When the packet size is 128 B (Figure 9(b)), with the number of flow rules growing, the forwarding rates of NMP with FAS and the commodity PC both vary little. And NMP with FAS is 18% higher than the commodity PC. In summary, the FAS can provide higher forwarding rate, especially for small-size packets.

We evaluate the forwarding latency with different packet sizes and different number of flow rules. As Figure 10 shows,



(a) Packet size = 64 B

(b) Packet size = 128 B

(c) Packet size = 256 B

(d) Packet size = 512 B

(e) Packet size = 1024 B

(f) Packet size = 1500 B

Figure 10: Comparison of forwarding latency with various packet sizes.

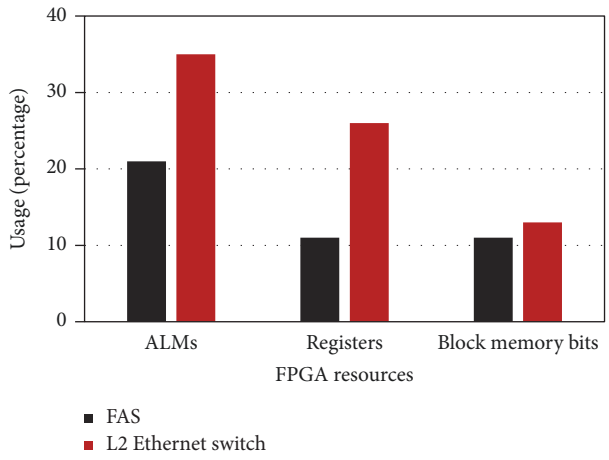


Figure 11: The FPGA resource usages of FAS and L2 Ethernet switch.

in the experiment, Skb forwarding indicates packet forwarding with the conversion operation between NMP packet and Skb data structure. It can be observed that the latency of Skb forwarding on NMP is close to the latency of the NMP packets. That means the cost of conversion operation between NMP packet and Skb is quite small. With the growing of packet size, the latency gap between Skb forwarding and Open vSwitch is decreasing. That is because if the packet size is small, there will be more packets in the software processing queue. The packets will experience longer delay when queuing. That incurs large forwarding latency.

At last, we compare the implementation complexity of FAS to a standard L2 Ethernet switch. The consumption of FPGA resource is depicted in Figure 11. As we can observe, the logic utilization of FAS in ALMs is about 40% less than the L2 Ethernet switch. The resource usage results suggest that the implementation of FAS is simple and feasible in FPGA.

We can conclude from the experiment that FAS provides considerable acceleration for OpenFlow software forwarding, especially for small packets. And the implementation complexity in FPGA is acceptable.

7. Conclusions

The SDN switches are the fundamental infrastructure to supply flexible control of flows. SDN software switches running on commodity multicore platforms are widely deployed due to their upgradability, programmability, and low cost. However, the forwarding performance as well as security capacity provided by general-purpose SDN software switch platform is usually not satisfied. The case becomes even worse for OpenFlow forwarding.

In this paper, we design and implement an FPGA-based mechanism accelerating and securing SDN software switches, namely, FAS. FAS provides a framework to offload the time-consuming modules and real-time security modules of SDN software switch and it employs some optimization techniques for solving the performance bottleneck between software and FPGA hardware. Our experimental results show that FAS utilizes reasonable FPGA resources and outperforms commodity platforms with nearly 44% higher forwarding rate for small packets. FAS can also be used to enhance the security of SDN software switches by allowing the bump-in-the-wire security modules to be integrated in FPGA.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was supported by a research grant from Chinese National Programs for Key Technology of Software Definition Network (SDN) Supporting Resource Elasticity Scheduling and Equipment Development (863 Programs) (no. 2015AA016103) and National Natural Science Foundation: Synergy Research on CPU/FPGA Heterogeneous Network Processing System for Complex Network Applications (no. 61702538). The authors thank Dr. Jianbiao Mao and other helpful friends for great help.

References

[1] *Software-Defined Networking: The New Norm for Networks, ONF White Paper*, http://www.opennetworking.org.

[2] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM CCR*, vol. 38, no. 2, pp. 69–74, 2008.

[3] *OpenFlow Specification 1.5.1, Open Networking Foundation*, 2015, http://www.opennetworking.org.

[4] P. Bosshart, G. Gibb, H.-S. Kim et al., "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *Proceedings of the ACM SIGCOMM 2013 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '13)*, pp. 99–110, August 2013.

[5] R. Ozdag, *Intel Ethernet Switch FM6000 Series Software Defined Networking*, August, Intel Corporation, 2012.

[6] The OpenFlow Consortium, *Openflow Switching Reference System*, January 2011, http://www.openflowswitch.org.

[7] "OFSoftSwitch13," http://cpqd.github.com/ofsoftswitch13.

[8] Y. Mundada, R. Sherwood, and N. Feamster, "An OpenFlow switch element for Click , in Symposium on Click Modular Router," 2009.

[9] *Open vSwitch – An Open Virtual Switch*, September 2014, http://www.openvswitch.org.

[10] Z. Cai, Z. Wang, K. Zheng, and J. Cao, "A Distributed TCAM coprocessor architecture for integrated longest prefix matching, policy filtering, and content filtering," *IEEE Transactions on Computers*, vol. 62, no. 3, pp. 417–427, 2013.

[11] A. Putnam, A. M. Caulfield, E. S. Chung et al., "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proceedings of the ACM/IEEE 41st International Symposium on Computer Architecture (ISCA '14)*, pp. 13–24, IEEE, Minneapolis, Minn, USA, June 2014.

[12] N. Shelly, E. J. Jackson, T. Koponen, N. McKeown, and J. Rajahalme, "Flow caching for high entropy packet fields," in *Proceedings of the 3rd ACM SIGCOMM 2014 Workshop on Hot Topics in Software Defined Networking (HotSDN '14)*, pp. 151–156, USA, August 2014.

[13] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an OpenFlow switch on the NetFPGA platform," in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '08)*, pp. 1–9, USA, November 2008.

[14] G. Pongrácz, L. Molnár, Z. L. Kis, and Z. Turányi, "Cheap silicon: A myth or reality? Picking the right data plane hardware for software defined networking," in *Proceedings of the 2013 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN '13)*, pp. 103–108, China, August 2013.

[15] B. Pfaff, J. Pettit, T. Koponen et al., "The design and implementation of open vSwitch," in *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI '15)*, pp. 117–130, USA, May 2015.

[16] A. Tirumala, F. Qin, J. Dugan et al., *iPerf: TCP/UDP Bandwidth Measurement Tool*, 2008.

[17] OProfile, http://oprofile.sourceforge.net.

[18] G. Liao, X. Znu, and L. Bnuyan, "A new server I/O architecture for high speed networks," in *Proceedings of the 17th International Symposium on High-Performance Computer Architecture (HPCA '11)*, pp. 255–265, USA, February 2011.

[19] Intel, "Intel data plane development kit (intel DPDK)," in *Programmer's Guide*, October 2013.

[20] L. Rizzo, "Netmap: A novel framework for fast packet I/O," in *Proceedings of the USENIX Annual Technical Conference*, pp. 101–112, 2012.

[21] K. Subramanian, L. D'Antoni, and A. Akella, "Genesis: Synthesizing forwarding tables in multi-tenant networks," in *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL '17)*, pp. 572–585, France, January 2017.

[22] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: A GPU-accelerated software router," in *Proceedings of the 7th International Conference on Autonomic Computing (SIGCOMM '10)*, pp. 195–206, India, September 2010.

[23] L. Tang, J. Yan, Z. Sun, T. Li, and M. Zhang, "Towards high-performance packet processing on commodity multi-cores: current issues and future directions," *Science China Information Sciences*, pp. 1–16, 2015.

[24] L. Rizzo, M. Carbone, and G. Catalli, "Transparent acceleration of software packet forwarding using netmap," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM '12)*, pp. 2471–2479, USA, March 2012.

[25] V. Tanyingyong, M. Hidell, and P. Sjodin, "Using hardware classification to improve PC-based OpenFlow switching," in *Proceedings of the 2011 IEEE 12th International Conference on High Performance Switching and Routing (HPSR '11)*, pp. 215–221, Spain, July 2011.

[26] R. Narayanan, S. Kotha, G. Lin et al., "Macroflows and microflows: Enabling rapid network innovation through a split SDN data plane," in *Proceedings of the 1st European Workshop on Software Defined Networks (EWSDN '12)*, pp. 79–84, Germany, October 2012.