A Kernel-Space POF Virtual Switch

Xiaoxiao Li[1,2], Yulei Wu[1,3], Jingguo Ge[1], Hongbo Zheng[1], Yuepeng E[1], Chunjing Han[1], Honglei Lv[1]

[1]State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, 100093, China

[2]University of Chinese Academy of Sciences, Beijing, 100049, China

[3]School of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter, EX4 4QF, UK

***Abstract*—Protocol Oblivious Forwarding (POF) aims at providing a standard southbound interface for sustainable Software Defined Networking (SDN) evolvement. It overcomes the limitations of popular OpenFlow protocols (an existing widely-adopted southbound interface), through the enhancement of SDN forwarding plane. This paper pioneers the design and implementation of a Kernel-space POF Virtual Switch (K_POFVS) on Linux platform. K_POFVS can improve the packet processing speed, through fast packet forwarding and the capability of adding/deleting/modifying protocol fields in kernel space. In addition, it is able to enhance flow table matching speed, by separating the mask table (consisting of flow entry masks used to figure out the matching field) and the flow table under a caching mechanism. Furthermore, K_POFVS can achieve efficient communication between the kernel space and the user space, via extending the Netlink communication between them. Experimental results show that K_POFVS can provide much better performance than existing user-space POF virtual switches, in terms of packet forwarding delay, packet processing delay and packet transmission rate.**

Keywords*—*Virtual Switch; Flow Table; OpenFlow; Protocol Oblivious Forwarding

1. Introduction

The Software Defined Networking (SDN) [1] is an emerging network architecture. It significantly reduces the diversity of hardware devices in the network through separating the control plane and the forwarding plane [2], thereby reducing the network complexity and capital expenditure (CAPEX). The logically centralized controller

in SDN can facilitate the network management, and, thus can reduce the network operational expenditure (OPEX). In addition, SDN is a promising architecture to foster the protocol innovations due to the shortened development and deployment lifecycle. The open SDN southbound interface breaks the traditional closed network equipment, facilitating users to dynamically adjust the network configuration according to their instant needs [3]. Among the many SDN reference architectures, OpenFlow as a popular southbound interface has been widely recognized by both academia and industry [4]. However, it faces several challenges with the development of OpenFlow. For example, the number of matching fields are increased from 12 to 41 as the introduction of new versions, and more matching fields would be needed for emerging and new applications [5]; several versions of OpenFlow protocols are not compatible with the others. The root cause of these problems is that OpenFlow cannot recognize the fields of unknown protocols, since it is a protocol-dependent southbound interface [6].

Protocol Oblivious Forwarding (POF) is an enhancement to OpenFlow-based SDN forwarding plane, which does not need to understand specific packet format. Thus, new network protocols and services can be supported without modifying the network device code [7]. To achieve this purpose, POF leverages {offset, length} data structure to define the protocol field based on its Flow Instruction Set (FIS). The combination of different instructions can perform parsing, editing and forwarding actions of data packets. FIS enables forwarding rules and forwarding device independent, improving the extensibility of forwarding plane, which truly realizes the separation of control plane and data forwarding plane [8]. With POF, users are able to quickly deploy customized applications in real-world networking environment and do not have to upgrade the hardware equipment.

Network Virtualization (NV) has become a popular topic in recent years, including a broad set of technologies like Virtual Local Area Networks (VLANs) [9]. The virtual switch is an important branch of NV, which connects the physical host and virtual machines. It also provides a large number of virtual network ports for virtual machines. The virtual switch can solve the link redundancy problem of traditional networks, reducing the hardware cost and

improving the utilization of network resources. The Open vSwitch (OVS), a virtual switch of OpenFlow, is the main software equipment of the forwarding plane for OpenFlow-based SDN deployment, in particular in the NV environment [10]. The OVS supports flexible forwarding strategy, improves the link and port utilization, simplifies the operation and maintenance complexity, shortens the design lifecycle and reduces the hardware cost. However, the limitations of OpenFlow also apply to OVS, resulting in the explosion of the number of matching fields in the flow table of OVS which can cause the network overload issues [11]. In addition, different versions of OpenFlow protocols have different flow table structures, increasing the maintenance cost of control plane in OVS.

The virtual switch of POF performs the data packet forwarding through FIS, so that it does not need to be aware of network protocols and forwarding processes. The existing POF virtual switch developed in the user space would produce several issues on the system performance, e.g., the increase in the time overhead when performing system function invocations. The kernel-space POF virtual switch is able to improve the system performance; however, its design possesses several challenges, including the function set partitioning problem, the contradiction between the high-speed flow table matching in the kernel space and the high communication cost between the user space and the kernel space, and the communication methods between the user space and the kernel space. These problems are essential in the design of a high-performance kernel space virtual switch for POF.

This paper targets at these challenging problems and pioneers the design and implementation of a Kernel-space POF Virtual Switch (K_POFVS) on Linux platform based on POF FIS. K_POFVS is able to provide the fast packet forwarding and the capability of adding/deleting/modifying protocol fields in the kernel space, improving the data packet processing speed. It can separate the mask table and the flow table based on a caching mechanism, enhancing the speed of flow table matching. Furthermore, K_POFVS extends the Netlink communication between the kernel space and the user space, achieving an efficient communication between them. A test environment is established in Ubuntu 14.04 Linux platform to evaluate the performance of the developed K_POFVS.

Experimental results show that K_POFVS can provide much better performance than existing POF virtual switches developed in the user space, in terms of packet forwarding delay, packet processing delay and packet transmission rate.

The rest of this paper is organized as follows. Section 2 introduces the related work on virtual switches. Section 3 presents the design and implementation of K_POFVS. The performance experiments and comparisons of K_POFVS are given in Section 4. Finally, Section 5 concludes this study.

2. Related Work

The virtual switch provides layer 2 network access and part of the layer 3 network functions for virtual machines. These functions can provide a large number of virtual network ports and high-speed online interfaces, so as to reduce the cost of equipment, improve the flexibility of network configuration, and enhance the utilization of network resources. The virtual switch can be implemented in the form of either hardware, e.g., Virtual Ethernet Port Aggregator (VEPA) technology [12] or software, e.g., OVS which is usually integrated with the virtual hypervisor to support users with flexible networking in Cloud computing.

The VEPA technical standard was initiated by many large commercial companies, e.g., HP and IBM. The core idea is to export the traffic from the virtual machine to the physical switch [13]. It realized the flow statistics, safety control and management functions. It transplanted the virtual switch function to the physical Network Interface Card (NIC) of servers. This solves the problem that virtual switch occupies the Central Processing Unit (CPU) resources which affects the performance of virtual machines. To use the Peripheral Component Interconnect Express (PCIe) NIC supporting the Single Root I/O Virtualization (SR-IOV) features, the virtual machine can directly access the PCIe NIC on the register and I/O queues. However, the traffic flow between virtual machines consumes much network bandwidth and increases the data transmission delay.

Cisco nexus 1000V (N1KV) was the virtual switch used in VMware ESX environment, supporting Cisco VN-

Link technology and providing a policy based virtual machine connection [14]. It included two modules: VEM and VSM. VEM was used to achieve the function of the virtual switch, and VSM was responsible for the management, monitoring and configuration of the virtual switch. N1KV provided VLAN, Quality-of-Service (QoS), NetFlow traffic control and other functions with a high availability and scalability. However, it required the use of vSphere Enterprise Plus 4.0 VMware or higher version, and achieves its maximized performance only when deployed in data centers with pure Cisco equipment.

OVS is based on the software implementation of OpenFlow switches, following the Apache 2.0 license and supporting a variety of standard management interfaces and protocols [15]. It is able to integrate with many virtualization platforms to provide flexible network interconnection between virtual machines [16]. OVS provides two forwarding data paths, one is based on the slow channel of user space, and the other one is based on the fast channel of Linux kernel space. Specifically, the ovs-vswitchd module of the user space is used to achieve the core functions of the OpenFlow switch, being responsible for the communication with the kernel module and the controller. However, the OpenFlow-based OVS cannot recognize the fields of unknown protocols. With the needs of protocol version upgrade to support new fields, the OVS software is required to be upgraded as well. The ever-increasing number of supporting fields in new protocol versions will result in heavy load problems in communication networks.

The POF virtual switch can get rid of the limitations of supporting specific protocols, and thus enhance the ability of control layer. The existing POF virtual switch was developed in Linux user space, decoupling the forwarding plane with the specific protocol. It performed the processing and forwarding of any protocol message through the installation of high efficient FIS by controllers [17]. Introducing flow metadata as global storage resources to store the flow data and state information makes the forwarding plane support the stateful packet processing and thus facilitate the management of the forwarding plane. However, there exists several issues in the

process of user space program execution, e.g., the process switching and task scheduling affect the performance of data forwarding.

To sum up, the virtual switch based on the hardware implementation needs specific NIC, producing high hardware cost and low flexibility. In contrast, the virtual switch based on software implementation has a short development lifecycle, low hardware cost and flexible network deployment. However, N1KV software virtual switch only supports VMware virtual machine, lacking the wide applicability. In addition, due to the limitations of OpenFlow protocols, the message processing capacity of OVS is limited. POF-based virtual switches can be used to effectively support new protocols. In comparison to the POF virtual switch developed in the user space, the kernel-space POF virtual switch can effectively perform system function invocations and improve its performance. In what follows, we will present the design and implementation of a high-performance kernel-space POF virtual switch.

3. Design and Implementation of Kernel-based POF Virtual Switches (K_POFVS)

3.1 POF Overview

POF can support any new protocols without modifying the code of forwarding devices. The protocol filed is defined using the data structure of {offset, length}, where the *offset* indicates the relative offset of the protocol field, and the *length* indicates the length of the protocol field.

Fig. 1 shows the reply packet of ARP protocol processing based on the POF architecture, where {96, 16} represents the Type field, {160, 16} denotes the opcode of ARP, and {304, 32} is the destination IP address [18]. The instruction is given based on the offset and length, so as not to rely on a specific protocol to perform the packet forwarding.

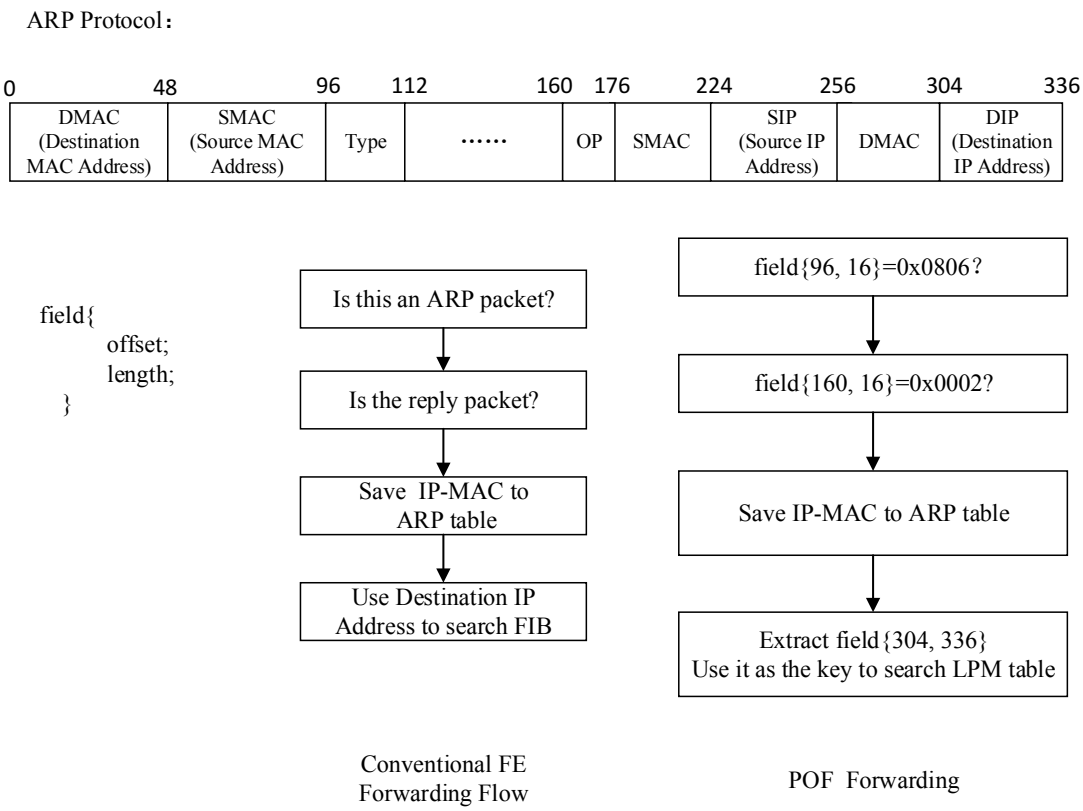

Figure 1. The POF forwarding process

3.2 Framework and Module Design of K_POFVS

The framework design of K_POFVS is shown in Fig. 2. K_ POFVS consists of the design in the user space and the kernel space. The *kernel space* stores a partial flow table and is responsible for fast packet forwarding, and the *user space* stores the whole flow table and is responsible for the communication with the kernel space and the control plane, and for processing the packets that are not successfully matched by the flow table in the kernel space. According to the procedure of packet processing, the main functional modules in the kernel space are Packet Receiving module, Key Value Extraction module, Flow Table Matching module and Instruction Execution module. The main functional modules in the user space are Packet Processing module, Communication module and Configuration module.

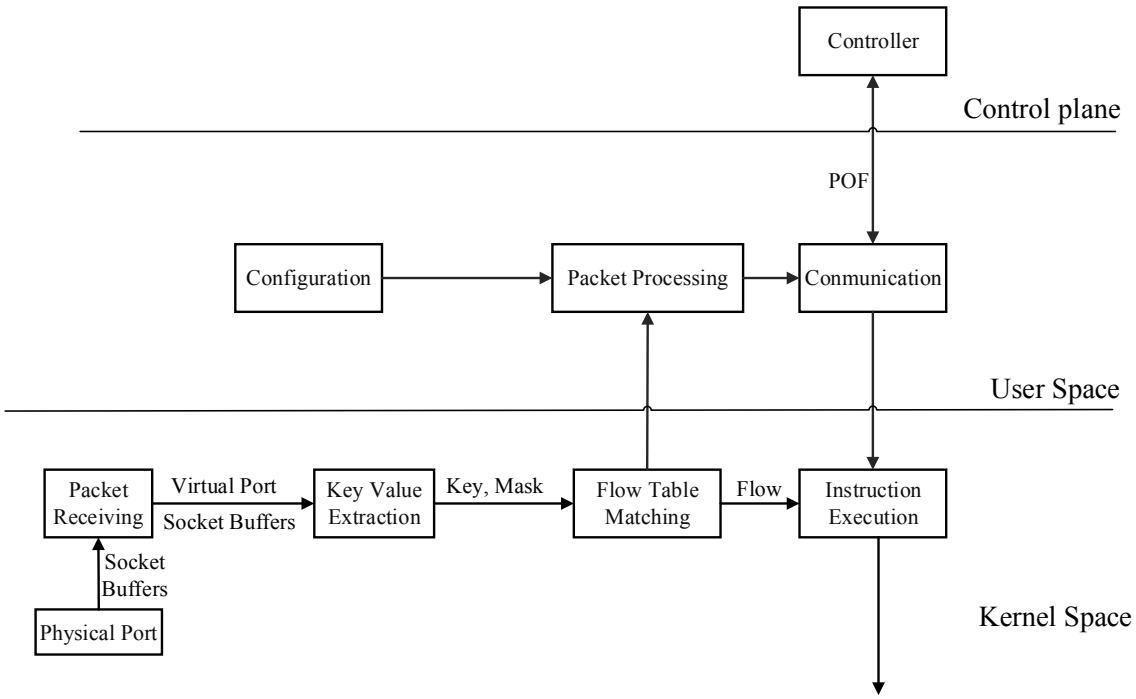

Figure 2. The design of POF framework

In the kernel space, Packet Receiving module is used for polling the port status, receiving packet and counting the status information of packets. Key Value Extraction module is responsible for extracting the partial information of packet as a key value, such as priority and port number, which is stored for subsequence matching. Flow Table Matching module is the most important module in the kernel space, which is used to get the position of a flow table through a series of masks calculated based on the key value. If the packet is matched successfully, it goes to the next module (i.e., Instruction Execution module), otherwise it will be submitted to the user space. Instruction Execution module is used to parse the instructions of the flow table and perform the packet processing. The instructions are performed by the POF FIS, including forwarding packets, setting values of protocol fields, adding protocol fields and calculating the checksum.

In the user space, Communication module is responsible for the communication between the user space and the kernel space, and also between the user space and the control plane. It is also used for the message parsing and transmission. Packet Processing module is used to receive the packets that are not successfully matched in the kernel space, and search the flow table in the user space. If the searching is successful, Communication module

distributes the matching flow entry to the kernel space and forward the packet, otherwise the packet needs to be sent to the Controller for processing. Configuration module is adopted for the configuration of the bridge, the port and the flow table.

3.3 The division of function set in K_POFVS

In the POF virtual switch developed in the user space (U_POFVS), packets should be copied and sent to the user space for processing. There are many system function invocations and process switching operations which affect the performance of packet forwarding in the virtual switch.

Since the kernel space resources are limited, it cannot achieve the full functionalities of the virtual switch. However, the network flow has temporal and spatial locality. *Temporal locality* refers to the case that the flow table to be matched at the current moment will still be matched at the next moment. *Spatial locality* refers to the case that if the entry of the flow table is matched at the moment, the entry of the flow table in other switches around will be matched at the next moment. Therefore, in this context we cache some flow entries that have been matched successfully in the user space into the Flow Table Matching module of kernel space [19]. When the packet with the same matching fields arrives, it can be forwarded directly through the kernel space and does not need to be sent to the user space, effectively reducing the number of system function invocations.

Based on the above analysis, the division of function set introduces the OVS design in the kernel space. The functions of packet receiving, key value extraction, flow table matching and instruction execution are implemented in the kernel space. It supports the modifying/adding/deleting protocol fields to achieve fast packet forwarding.

3.4 The implementation of K_POFVS

The implementation of K_POFVS faces several challenges, such as the migration of instructions in the user space to the kernel space, and the contradiction between the high-speed flow table matching in the kernel space and the high communication cost between the user space and the kernel space. These issues need to be solved in

the design of high-performance kernel space virtual switches.

3.4.1 The related data structures

In the process of the packet matching and forwarding in the kernel space, we propose a set of main data structures: pof_flow_key, k_pof_match and the data structure for instruction execution like k_pof_action_set_field.

```
Struct pof_flow_key{
   u16 flow_len;                    /* length of the key*/
   u8 flow[OVS_POF_FLOW_SIZE];      /* value of the key*/
}

struct k_pof_match{
   uint16_t field_id;    /*0xffff means the match field comes from metadata,
                         0x8XXX means the match field comes from table
                         parameter,
                         otherwise means the match field comes from packet*/
   uint16_t offset;      /*the relative offset to its packet/metadata header
                         in unit of bit*/
   uint16_t len;         /*length of the match field in unit of bit*/
   uint8_t pad[2];       /*8 bytes aligned*/
}

struct k_pof_action_set_field{
   uint16_t type;        /*type of the action, such as set, add or delete*/
   uint16_t len;         /*length of this struct in bytes*/
   k_pof_match_x  field_setting; /*the protocol field which needs to be set*/
}
```

3.4.2 The process of flow table matching

The protocol in POF can contain many user-defined fields, represented by the starting position of the field and its length. For example, IPv4 is a protocol, including the fields of source address, destination address, and quality-of-service. The flow table is composed of several protocol fields with the corresponding execution instructions which are used for packet matching and processing. The protocol fields contain the values of each field and the flow table mask. The execution instructions are a set of instructions.

Different fields of a protocol can have different combinations to represent many flow entries in a flow table,

but the mask of these flow entries may be the same. In K_POFVS, multi-segment masks corresponding to one flow entry are integrated into one mask record, which is stored separately with the flow table as a new Mask Table (MT). When the packet goes into K_POFVS, it first extracts the key value and performs the 'AND' operation with a record in the MT, and then goes for the matching process. It can effectively reduce the number of times required for mask matching in the protocol field, thereby reducing the total time of mask matching.

The flow table matching process is shown in Fig. 3. When the packet arrives at K_POFVS, its header information is extracted and stored in the array, flow[], to form the unmasked key. K_POFVS then uses the unmasked key to do 'AND' operation with a record in the MT to generate the mask key. After that, it leverages the hash algorithm on the mask key, and find the right flow entry according to the hash result.

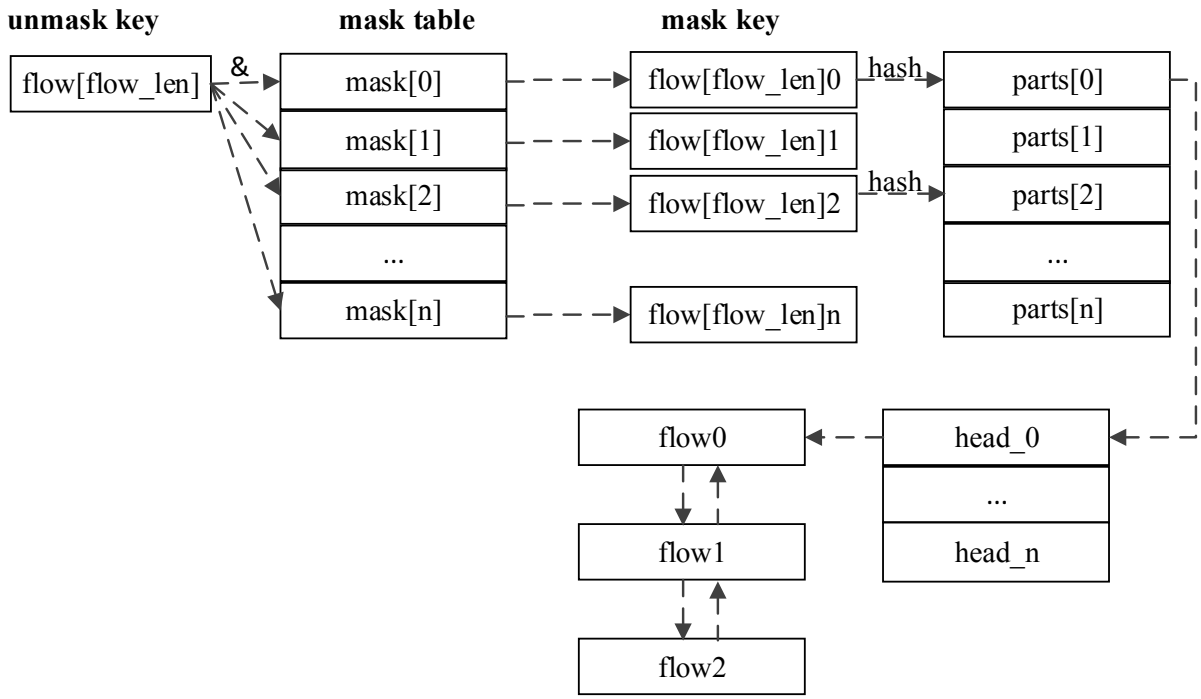

Figure 3. The process of flow table matching

3.4.3 The extension of communication functions

K_POFVS has the issue in the communication between the kernel space and the user space. There are several mechanisms of data exchange between these two spaces in Linux, such as system call, procfs, relayfs and Netlink.

System call is initiated by the user space. Procfs and relayfs communication mechanisms are based on the file system. Those are all simple communication mechanisms, which cannot meet the two-way data switching between the kernel space and the user space. Netlink facilitates the communication by providing a set of APIs for the kernel space and a set of sockets for the user space. It is an asynchronous communication mechanism and supports multicast. Since the socket itself possesses the characteristics of two-way communication, non-blocking and emergency, it can therefore meet the needs of two-way data exchange.

Since the Netlink communication mechanism does not have the POF-related fields, it cannot perform the POF packet parsing. Therefore, in this paper we add new attribute definitions associated with POF in the Netlink communication mechanism (see Fig. 4) to make the communication between the user space and the kernel space successful in K_POFVS. Specifically, we add eight attributes, four for key_attr and the other four attributes for action_attr, shown in Fig. 4.

key_attr	**action_attr**	
ovs_key_attr_priority	ovs_action_attr_unspec	existing definitions
ovs_key_attr_in_port	ovs_action_attr_output	
ovs_key_attr_ipv4	ovs_action_attr_set	
ovs_key_attr_tcp	ovs_action_attr_hash	
ovs_key_attr_udp	ovs_key_attr_userspace	
……	……	
pof_key_attr_pof_field	pof_action_attr_pof_set	added definitions
pof_key_attr_pof_value	pof_action_attr_pof_add_field	
pof_key_attr_pof_mask	pof_action_attr_pof_calc_checsum	
pof_key_attr_pof_field_len	pof_action_attr_pof_drop	

Figure 4. The Netlink mechanism in K_POFVS

The pof_key_attr_pof_value, pof_key_attr_pof_filed, pof_key_attr_pof_mask and pof_key_attr_pof_field_len allow the user space to parse the message when getting it. The pof_action_attr_pof_set, pof_action_attr_pof_add_field, pof_action_attr_pof_calc_checksum and pof_action_attr_pof_drop are used to

execute the instructions on packets. When the packet is not matched in the kernel space, it will send the upcall information to the user space which includes the port number and key value. The user space will parse the packets according to the Netlink and search the flow table. If the search is successful, Communication module sends the packets to the kernel space to execute the corresponding instructions. If the lookup fails, it will send a packet-in message to the Controller.

According to the test results of existing POF virtual switches and OVS, we adopt separate functions in the user space and the kernel space. In particular, the flow table forwarding functionality is implemented in the kernel space, taking the advantage of the effectiveness of kernel space to store some flow tables and mask tables that have been matched in order to benefit future flow table lookups. This can significantly enhance the performance of packet processing and forwarding in the kernel-space POF virtual switch.

4. Performance Measurement and Analysis

Extensive experimental results are conducted to evaluate the performance of the developed K_POFVS. In particular, we compare the performance of K_POFVS with that of the existing U_POFVS in terms of several key performance metrics to show its relative merits.

The experimental environment in a PC is depicted in Fig. 5, with the system configuration shown in Table 1. The developed K_POFVS and the existing U_POFVS are installed in the *virtual switch*. The virtual switch establishes bridge br0, and adds both virtual Ethernet 1 (ve1) and ve3 to br0. ve2 and ve4 are isolated by the namespace, and are assigned the same segment of IP addresses, where ve2 is in a *client* and ve4 is in a *server*. Two virtual Ethernet interface (veth) pairs, i.e., ve1-ve2 and ve3-ve4 are created to allow the packages generated by the client to go through the virtual switch to reach the server. The controller is responsible for generating the flow table rules based on the real network traffic. Typical rules are shown in Table 2.

Table 1. The system configuration of Virtual Switch

System	Ubuntu 14.04
RAM	8G
CPU	Intel core i5 (4 cores)
HDD	150G

Table 2. List of example rules in the flow table

Regulation	Protocol field: value.mask		Action	Function
1	type:0806.ffff		output(ve1, ve3)	broadcast the ARP packet to every port
2	type:0800.ffff	dstIP:c0a8ff02.ffffffff	output(ve1)	send packets to port ve1 if DstIP is c0a8ff02
3	type:0800.ffff	dstIP:c0a8ff01.ffffffff	output(ve3)	send packets to port ve3 if DstIP is c0a8ff01

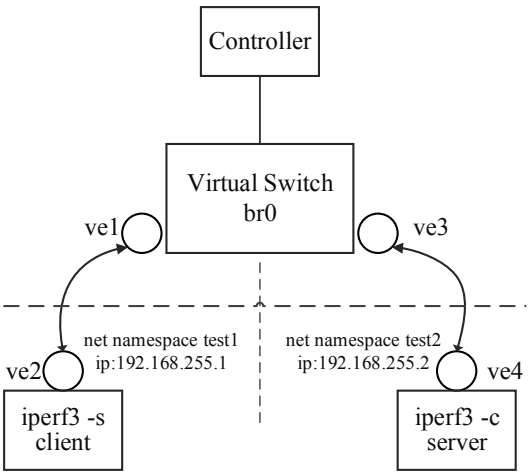

Figure 5. The test network environment

According to the standard of measuring network performance, the iperf3 tool with the fixed Maximum Segment Size (MSS) are used in the server and the client for the performance comparisons of K_POFVS and U_POFVS. Three key performance metrics: packet processing delay, packet forwarding delay and packet transmission rate are used for the evaluation and analysis in this section. The packet processing delay is defined as the duration from the time that the packet is received by ve1 to the time that the packet is sent out by ve3. The packet forwarding delay is defined as the duration from the time that the packet is received by ve2, after traversing the path ve1-ve3-ve4-ve3-ve1, to the time that the packet is returned to ve2. The packet transmission rate is defined as the total bytes of the packets sent per second.

4.1 Average packet processing delay

In this section, we use both the real traffic and the synthetic traffic to evaluate the packet processing time of K_POFVS and U_POFVS virtual switches. The SystemTap tool [20] with a script written by ourselves is used to obtain the packet processing time. The packet processing time in K_POFVS includes the time for receiving data packets from the network card, matching the flow table and taking the required actions. The packet processing time in U_POFVS includes the time for receiving data from the network card, switching between the user space and the kernel space, scheduling in the user space, matching the flow table and taking the required actions.

4.1.1 Flow collection and extraction

TCPDUMP is used to collect real traffic from the server and save it as .CAP file, and Scapy [21] is used to generate synthetic traffic. The real traffic is collected in a manner that each CAP file either has 2000 packets or 10000 packets. The fields of each flow, such as IP address, protocol type and port number are extracted. The duplicate flows are removed in order to generate the rules of flow table.

4.1.2 Flow table generation

To generate the flow table, a test_nflows() function is added in the controller (POX controller [22] in this deployment). The pseudo-code of test_nflows() function is given below.

```
Function test_nflows(event):
    protocol definition
        ofmatch <- fieldId
        ofmatch <- offset
    open traffic file
    add flow table
        flowtable <- tableType
        flowtable <- tableId
        ……
    add match field
        tempmatchx <- fieldId
        tempmatchx <- offset
        tempmatchx <- length
    for eachline in file to EOF
        tempmatchx <- length
        tempmatchx <- set_value
        tempmatchx <- set_mask
        ……
        add instruction applyaction
```

```
        action <- actionType
        action <- portId
        ……
end
send flow table
```

4.1.3 The performance results

It is important to compare the packet processing delay by K_POFVS and U_POFVS to quantitatively show the relative merits of K_POFVS.

Table 3 shows the packet processing delay by K_POFVS and U_POFVS for the cases of 2000 packets and 10000 packets, respectively, under both the real traffic and the synthetic traffic. From the results of real traffic, we can find that K_POFVS reduces the packet processing delay by about 61.49% and 58.84% than U_POFVS for the cases of 2000 packets and 10000 packets, respectively. In addition, we can observe that the packet processing delay under the real traffic is lower than that under the synthetic traffic in both the two virtual switches. That is because in the real network environment, most traffic has high locality (see Section 3.3), resulting in high probability in flow entry matching which ultimately leads to lower packet processing delay under real traffic scenario.

Table 3. The packet processing time of two virtual switches

	K_POFVS(μs)		U_POFVS(μs)	
Number of packets	2000 packets	10000 packets	2000 packets	10000 packets
Real traffic	35.187	53.436	91.375	129.834
Synthetic traffic	57.59	75.629	164.014	205.379

4.2 Packet transmission rate and packet forwarding delay

In this section, we evaluate the packet transmission rate and packet forwarding delay under K_POFVS and U_POFVS. RFC2544 provides international standard for the evaluation of network interconnect devices, recommending the test frame length of Ethernet set to be 64, 128, 256, 512, 1024, 1280 and 1518 bytes. The corresponding MSS values with these frame length settings are shown in Table 4. The frame lengths of Ethernet

of 256 and 1518 bytes are used in this section for performance analysis.

Table 4. Frame length settings with corresponding MSS values

ping Delay Test	iperf3 Bandwidth Test
Frame Length (byte)	MSS (byte)
64	6
128	70
256	198
512	454
1024	966
1280	1222
1518	1460

Figs. 6 and 7 show the packet transmission rate under K_POFVS and U_POFVS when the MSS is set to be 198 bytes (small packet) and 1460 bytes (large packet), respectively. The results show that no matter how many flow entries are, the packet transmission rate under K_POFVS is higher than U_POFVS for both the small packet and large packet cases. That is because the packet processing of K_POFVS is faster than U_POFVS, leading to the higher packet transmission rate.

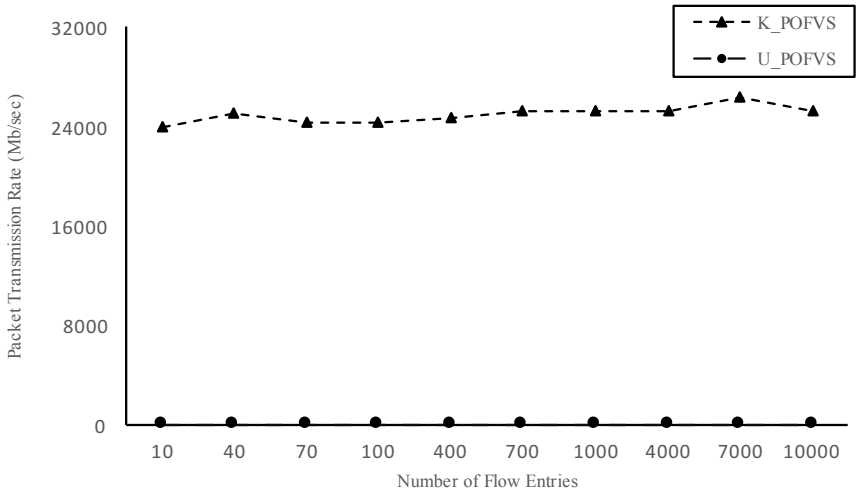

Figure 6. Packet transmission rate under K_POFVS and U_POFVS with different number of the flow entries and the MSS set to be 198 bytes.

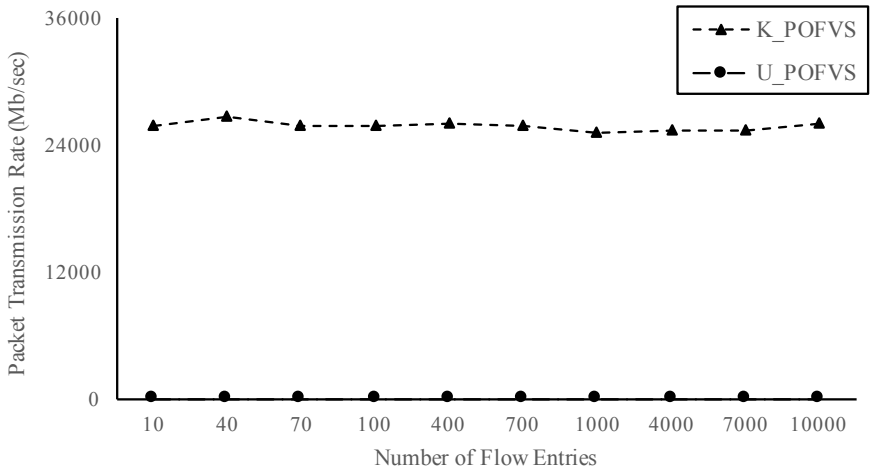

Figure 7. Packet transmission rate under K_POFVS and U_POFVS with different number of the flow entries and the MSS set to be 1460 bytes.

Figs. 8 and 9 depict the packet processing delay for K_POFVS and U_POFVS when the frame length is set to be 256 bytes (small packet) and 1518 bytes (large packet), respectively. From Fig. 8, we can obtain that the average packet forwarding delay is about 0.0684 ms under K_POFVS and 0.3509 ms under U_POFVS. Thus, K_POFVS decreases the packet forwarding delay by about 80.50% than U_POFVS under the small packet scenario. The result of large packet is similar to that of the small packet, where the average packet forwarding delay shown in Fig. 9 is about 0.0806 ms under K_POFVS and 0.6076 ms under U_POFVS; K_POFVS reduces the packet forwarding delay by about 86.73% compared to U_POFVS. The results demonstrate that the packet forwarding delay of K_POFVS is much lower than U_POFVS under different number of flow entries for both the small packet and large packet cases. That is because in K_POFVS, some flow entries are cached in the kernel space after the first packet is matched, resulting in higher processing speed than U_POFVS.

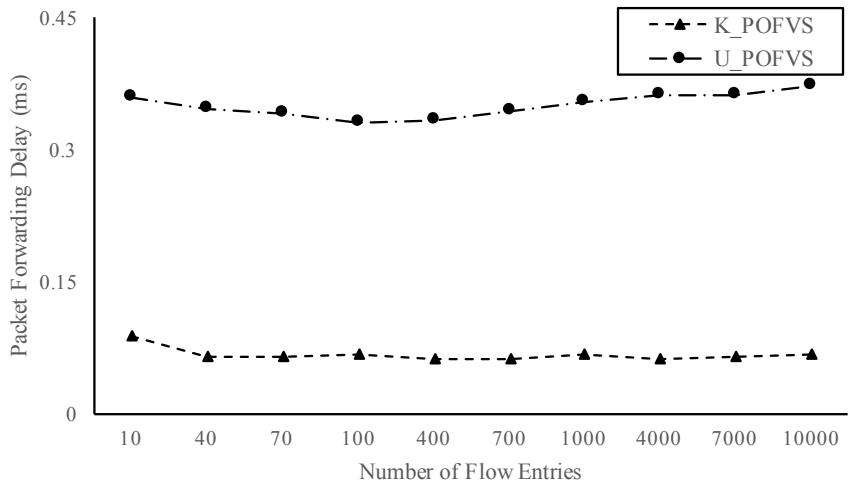

Figure 8. Packet forwarding delay under K_POFVS and U_POFVS with different number of flow entries and the frame length set to be 256 bytes.

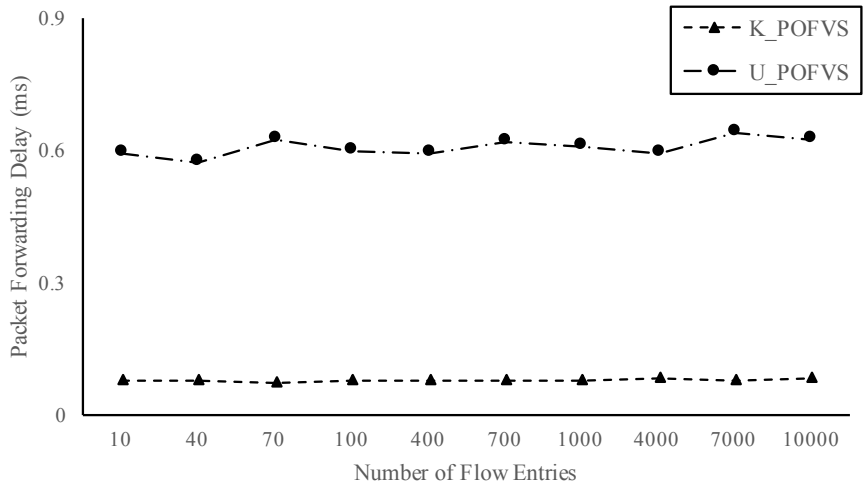

Figure 9. Packet forwarding delay under K_POFVS and U_POFVS with different number of flow entries and the frame length set to be 1518 bytes.

In summary, after implementing the POF virtual switch in the kernel space (K_POFVS) and the user space (U_POFVS), we found it is useful to cache partial flow tables in the kernel space. It can help to perform fast packet processing, reduce the number of times of re-directing packets to the user space, and improve the performance of the POF virtual switch.

5. Conclusions

Protocol Oblivious Forwarding (POF) can readily extend the application scenarios of OpenFlow-based SDN architecture. It provides a stable and future-proof forwarding plane, truly realizing the separation of control plane and data forwarding plane. POF does not need to understand the packet format in the forwarding plane, and can support any existing and emerging protocols. Due to the important role of virtual switch, this paper has designed and implemented a Kernel-space POF Virtual Switch (K_POFVS) in a Linux platform based on the POF Flow Instruction Set. The developed K_POFVS can achieve fast packet forwarding via caching partial flow tables in the kernel space, perform efficient flow table matching via separating the mask of flow entries with the flow table as a new mask table, and reduce the overhead for the communication between the user space and the kernel space through adding POF-related attributes in the Netlink communication mechanism. To show the merits of K_POFVS, we have compared its performance with that of existing POF virtual switches developed in the user space (U_POFVS). Extensive experimental results have demonstrated that, with given system parameter settings, K_POFVS can reduce the packet processing delay by about 60% and decrease the packet forwarding delay by about 80% in comparison with those of U_POFVS. The results have also shown that K_POFVS is able to provide a great improvement on the performance of POF virtual switches in terms of packet transmission rate.

Acknowledgements

This work is partially supported by the National Program on Key Basic Research Project of China (973 Program) under Grant No. 2012CB315803, the Strategic Priority Research Program of the Chinese Academy of Sciences under grant No. XDA06010306, the National Natural Science Foundation of China under Grant No. 61303241, and the University of Exeter's Innovation Platform – Link Fund under Award No. LF207.

References

[1] N. Mckeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner,

"OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69-74, 2008.

[2] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-defined networking: a comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 10-13, 2014.

[3] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 3, pp. 1617−1634, 2014.

[4] F. Hu, Q. Hao and K. Bao, "A survey on software-defined network and openflow: from concept to implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181-2206, 2014.

[5] H. Li, M. Dong and K. Ota, "Radio access network virtualization for the social Internet of Things," *IEEE Cloud Computing*, vol. 2, no. 6, pp. 42-50, 2015.

[6] H. Song, "Protocol-oblivious forwarding: unleash the power of SDN through a future-proof forwarding plane," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking (HotSDN'13)*, pp. 127-132, 2013.

[7] J. Yu, X. Wang, J. Song, Y. Zheng and H. Song, "Forwarding programming in protocol-oblivious instruction set," in *Proceedings of the 2014 IEEE 22nd International Conference on Network Protocols (ICNP'14)*, pp. 577-582, 2014.

[8] H. Song, J. Gong, H. Chen and J. Dustzadeh, "Unified POF programming for diversified SDN data plane," Eprint Arxiv, arXiv:1405.0060, 2014.

[9] N. M. Mosharaf Kabir Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862-876, 2010.

[10] B. Han, V. Gopalakrishnan, L. Ji and S.Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90-97, 2015.

[11] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll and P. Tran-Gia, "Modeling and performance evaluation of an OpenFlow architecture," in *Proceedings of the 23rd International Teletraffic Congress*, pp. 1-7, 2011.

[12] H. M. Tseng, H. L. Lee, J. W. Hu，T. L. Liu, J. G. Chang and W. C. Huang, "Network virtualization with cloud virtual switch," in *Proceedings of 2011 IEEE 17th International Conference on Parallel and Distributed Systems*, pp. 998-1003, 2011.

[13] J. Pettit, J. Gross, B. Pfaff, M. Casado and S. Crosby, "Virtual switching in an era of advanced edges," in

Proceedings of 2nd Workshop on Data Center Converged and Virtual Ethernet Switching (DC-CAVES'10), 2010.

[14] Cisco Nexus 1000V, Cisco Nexus 1000V Inter Cloud, http://www.cisco.com/web/CN/products/products_netsol/switches/; [accessed 16.01.15]

[15] J. Pettit, "Open vSwitch and the Intelligent Edge," *OpenStack Summit*, 2014.

[16] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wan, J. Stringer, P. Shelar, K. Amidon and M. Casado, "The Design and Implementation of Open vSwitch," in *Proceedings of 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI'15)*, pp. 117-130, 2015.

[17] POFSwitch, http://www.poforwarding.org

[18] Principle and Implementation of POF, http://www.poforwarding.org

[19] M. Dong, H. Li, K. Ota and J. Xiao, "Rule caching in SDN-enabled mobile access networks," *IEEE Network*, vol. 29, no. 4, pp. 40-45, 2015.

[20] Systemtap tutorial, https://sourceware.org/systemtap

[21] Scapy documentation, http://www.secdev.org/projects/scapy/doc

[22] POX Wiki, https://openflow.stanford.edu/display/ONL/POX+Wiki