Computer Networks 110 (2016) 91–103

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Efficient mismatched packet buffer management with packet order-preserving for OpenFlow networks

Jianbiao Mao, Biao Han*, Zhigang Sun, Xicheng Lu, Ziwen Zhang

College of Computer, National University of Defense Technology, China

ARTICLE INFO

Article history:
Received 25 January 2016
Revised 9 September 2016
Accepted 20 September 2016
Available online 21 September 2016

Keywords:
OpenFlow
Software-Defined Networking
Flow-granularity
Mismatched packet buffer management
Packet order-preserving

ABSTRACT

OpenFlow-based networks simplify network management and improve network programmability by centralized network control. Existing OpenFlow networks employ packet-granularity mismatched packet buffer management to reduce the switch-controller communication overhead. However, the impact of packet buffer management granularity on network performance has not been well investigated yet. In this paper, we propose a novel design of flow-granularity mismatched packet buffer management model for OpenFlow networks, which outperforms the existing packet-granularity buffer management approaches with less communication overhead between switches and controllers. By designing the flow action preprocessing mechanism, we prevent per-flow packet disorder with less packet drop ratio. We evaluate the performance of the proposed flow-granularity mismatched packet buffer management scheme by building prototypes on both software and hardware switches. Experimental results reveal that our proposed method can effectively reduce the switches-controllers communication overhead, as well as preserve packet order for multi-flow OpenFlow networks.

1. Introduction

Software Defined Networking (SDN) is a transforming networking design that simplifies network management and improves programmability of network. It separates the control plane from the data plane by moving the network intelligence to the logically centralized controller. Thus, SDN provides new abilities to enhance the flow scheduling, automatic network configuration, load balancing and network virtualization. It has been widely studied [1–3] and deployed in many scenarios such as datacenter networks [4,5]. OpenFlow is the standard southbound interface in SDN, which realizes packet forwarding based on network flows [6]. As a dumb and unintelligent network device, the OpenFlow switch invokes all flow rules from the intelligent controller. Although SDN brings greater network resource utilization and quality of service guarantees, the logically centralized controller becomes the bottleneck of the network performance.

In order to avoid the centralized controller become the performance bottleneck of the network, many works have been conducted to improve the capability and scalability of OpenFlow controllers [2,7–11]. Along with their efforts, buffering packets in OpenFlow switches is becoming a promising solution to reduce the communication overhead between switches and controllers. As pointed out by OpenFlow Specification 1.4 [6], when there is a buffer in the OpenFlow switch, instead of sending the whole packet to the controller to request for a forwarding rule, it only needs to send the digest of packet with the buffer address where the packet stores. Thus, the buffer reduces the size of *Packet-in* messages, thereby decreasing the communication overhead in OpenFlow channel.

However, as far as we know, current OpenFlow switches employ packet-granularity model for the mismatched packet buffer design [12,13]. Under this model, due to the communication delay between controllers and switches during a flow setup, a large number of packets belonged to a same flow will be sent to the controller as *Packet-in* messages, which will significantly degrade the network performance. In this paper, we consider a more efficient and coarse-grained mismatched packet buffer management design for OpenFlow networks. We focus on the problem of reducing the communication overhead between OpenFlow switches and controllers, either for UDP or TCP traffic, thus to improve the scalability of the centralized OpenFlow controllers. Connectionless UDP flows may induce burst *Packet-in* messages to controllers, which has been pointed out in [14]. On the other hand, for connection-oriented TCP flows, as forwarding rules have been installed in the flow table during the three-way handshake process, TCP traffic will not induce burst *Packet-in* messages to controllers. However, in networks with large dynamic traffic such as

* Corresponding author.
E-mail address: nudtbill@nudt.edu.cn (B. Han).

http://dx.doi.org/10.1016/j.comnet.2016.09.016
1389-1286/

datacenters, the rules in switches for already built TCP connections may be replaced frequently, which will lead to significant flow table look-up miss for a large number of TCP packets [15]. Therefore, connection-oriented TCP flows may also generate extra workloads from OpenFlow switches to controllers, which is similar as UDP. We will discuss the details of the packet-granularity buffer model in Section 2.2.

Besides, in order to prevent generating more *Packet-in* messages during the flow setup process, current OpenFlow switches installs flow rules first and then releases the mismatched packets in the buffer, which leads to per-flow packet disordering. When arriving at receiver, those out-of-order packets have to be reordered before they are sent to the application layer, which brings extra overhead for transmission. Especially for TCP flows, packet disordering would significantly decrease the performance of communication [16]. In this paper, to further reduce the transmission workload in OpenFlow channel and to achieve order-preserving transmission during flow setup, we propose an efficient flow-granularity mismatched packet buffer model with packet order-preserving design, named FPB, which is an enhancement to the OpenFlow control plane. In FPB, we buffer the mismatched packets in flow-granularity while only the first one of them is sent to the OpenFlow controller. By designing the flow action pre-processing mechanism, we prevent per-flow packet disorder with less packet drop ratio. We validate the efficiency of the proposed model by building prototypes on a software switch OFSoftSwtich [12] and a hardware switch based on NetMagic platform [17], which provide a potential reference design for the future OpenFlow networks.

The contributions of this paper can be summarized as follows:

- We propose a novel design of flow-granularity mismatched packet buffer management model for Openflow-based networks, which outperforms the existing packet-granularity buffer management approaches with less communication overhead between switches and controllers.
- By designing the mechanism of flow action pre-processing, we prevent per-flow packet disorder while reducing packet dropping in Openflow-based networks by handling the mismatched packet buffer.
- We evaluate the efficiency and performance of the proposed model by building prototypes on both software switch and hardware switches. The experimental results reveal that our proposed method can reduce the communication overhead between switches and controllers, as well as preserve packet order for multiple flows in Openflow-based networks.

The remainder of the paper is organized as follows. We describe the problem of buffering mismatched packet under the environment of datacenter networks in Section 2. In Section 3, we describe the processing procedure of our proposed buffer design and the comparison with other ways. In Section 4, we describe detailed design inside our buffer. In Section 5, we build prototypes both on software switch and hardware switch. Evaluation results show less workload on switches and controllers with our proposed FPB. Related works are introduced in Section 6. We conclude the paper and point out future work in Section 7.

2. Problem description

2.1. Reactive and proactive path setup mode

In OpenFlow networks, the controller adopts two approaches to build forwarding paths among hosts: proactive and reactive [18]. Time sensitive UDP flows are treated under the proactive mode in Openflow-enabled networks, in which the forwarding rules are pre-installed in the switches before the packets arrive. Therefore, it is efficient to greatly reduce the forwarding delay, even if it is suffering large packet drop ratio. In this paper, we mainly focus on the reactive mode in Openflow-enabled networks with burst mismatched packets, e.g., Openflow-based datacenter networks, where the traffic fluctuates much more frequently than other Openflow-based networks. Although installing rules reactively can harm the performance for OpenFlow networks, it is necessary and essential for network security policy and management. OpenFlow-based datacenter networks adopt the reactive way of “One-way Flow-setup” to set up the forwarding paths passively for some new flows [1–3]. As shown in Fig. 1, the source host A sends a new packet flow to the destination B after forwarding along n-hops of OpenFlow switches, which are noted as S1, S2,...,Sn. The process of reactive One-way Flow-setup mode is described as follows:

Step 1: When arriving at the first-hop OpenFlow switch, the first packet (P1) of the new flow encounters a flow-table miss. Then the control plane of the OpenFlow switch encapsulates the mismatched packet into a $Pkt-in$ message and sends it to the centralized controller through the established OpenFlow channel between the controller and the switch.

Step 2: After processing the Packet-in message, the controller sets up a bidirectional forwarding path between A and B by sending Flow-mod messages ($Flow-mod1$, $Flow-mod2$,!‘, $Flow-modn$) to install rules to corresponding switches (S1, S2,...,Sn) along the path.

Step 3: To avoid packet loss of P1, the controller sends a $Pkt-out$ message to S1 to forward P1 to the next-hop switch. Then P1 transmits along the already built forwarding path to B.

In this way, a flow only invokes the controller once at the edge of datacenter network (namely the first-hop switch) to install all rules for the flow. It minimizes the cost of flow setup to the centralized controller. Given this benefit of “One-way Flow-setup”, many existing SDN controllers (e.g., Floodlight, POX) use this way by default to build forwarding paths for flows currently. Besides, the Retransmit Time Out (RTO) in Linux OS is 200 ms by default, While the median latency for invoking a rule is approximate 3.95 ms [19], which is two orders less than RTO. Therefore, this added delay seldom cause packet retransmission in the reactive flow setup situations.

However, due to different delays between the controller and switches, the packet may not matched the flow table in latter switches. As described by Canini et al. [20], when P1 arrives at S2, the rule within $Flow-mod2$ has not been installed into S2. So P1 would be sent to the controller again to trigger rules. The controller that implicitly expects to see just one packet during the flow setup may behave incorrectly when multiple arrive. It leads to inconsistency of control logic for the flow, which should be avoided.

For analyzing the condition of “One-way Flow-setup”, we define the notation in Table 1. In order to guarantee that P1 would not miss the flow tables in subsequent switches after leaving the first-hop switch, the related transmission time should satisfy the

Table 1
Notations used in this paper.

n	Number of switches along the forwarding path
B_w	Bandwidth of flow
M	Max length(in Bytes) of a packet in datacenter network
M_{pi}	Length(in Bytes) of a Packet-in message with Buffer_id and packet digest
t_{flow}	Duration time of flow
$d_{A\rightarrow S1}$	Packet transmission delay from A to S1
$d_{S1\rightarrow C}$	Delay for generating Packet-in from S1 to C
d_C	Delay for processing packet in controller
$d_{C\rightarrow Si}$	Delay for rule install in Si from C,i = 2,3,...,n
$d_{S1\rightarrow Si}$	Packet transmission delay from S1 to Si,i = 2,3,...,n
$D1$	$d_{S1\rightarrow C}+d_C+d_{C\rightarrow S1}$

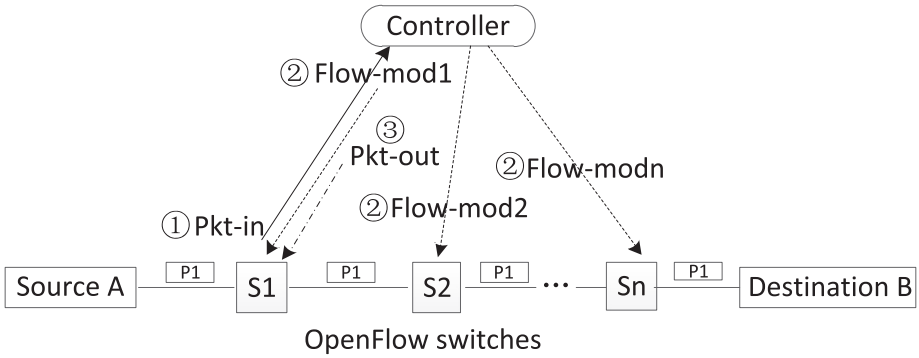


Fig. 1. Reactive One-way Flow-setup mode in a sample OpenFlow network.

inequation as follows:

$$d_{C\to S1} + d_{S1\to Si} > d_{C\to Si}, i = 2, 3, ..., n. \quad (1)$$

To meet the inequation, the centralized controller installs rules by sending Flow-mod messages to switches in the path reversely [20]. More precisely, the controller reverse the order by allowing switch Sn to install its rule first, followed by switch Sn−1, and the last to S1. Most current studies about OpenFlow-based datacenter networks implicitly ground on this inequation, which reduces the overhead of flow setup. Therefore, in this paper, we only analyze the problem of flow buffer for mismatched packets at the first-hop OpenFlow switch, which simplify the model to the edge of networks. As described above, there is a time delay between the first packet mismatching the rules and the edge switch installing the corresponding rule, so many subsequent packets in the same flow are also sent to the controller as Packet-in messages. As a result, the controller becomes overloaded and unstable.

2.2. Problem of existing packet buffer models

Here we present the methods of existing buffer model, including no-buffer model and packet-granularity Packet-in buffer model, for burst mismatched packets and point out their limitations. After the controller establishes the OpenFlow Channel with the switch, it sends an OFPT_FEATURES_REQUEST message to get the buffer capability of the switch. We first consider the switch with no buffer. The procedure of flow setup is shown in Fig. 2(a). P1, P2, P3 and P4 are in the same new flow sent from A to B. When the first packet P1 arrives at the switch, the switch has no rules for the packet and generates a Packet-in message including the whole packet. The controller responds the request with one Flow-mod message and one Packet-out message. The Flow-mod message contains the forwarding rule R_{A-B} for the flow, which consists of the match field $Match_{A-B}$ and the action field $Action_{A-B}$. The Flow-mod message contains the whole packet and the action $Action_{A-B}$. After the rule is installed to the flow table of the switch, packets in the flow will hit the rule and be executed with corresponding actions. However, due to the delay between the Packet-in sent and the rule installation, subsequent packets of the flow are also forwarded to the controllers, as P2 and P3 in the Fig. 2(a). Assuming that the controller is optimized, it identifies that P2 and P3 are in the same flow with P1 and does not send Flow-mod messages anymore. To avoid the packet loss, the controller also sends Packet-out messages including P2 and P3 respectively. To the common case with the notation in Table 1, the request number of Packet-in messages in a new flow setup with bandwidth B_w is as follows.

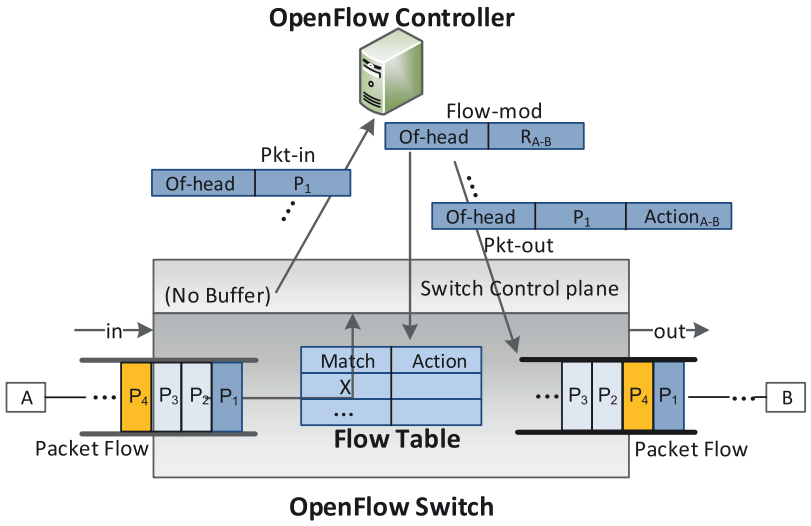
$$R_{nobuffer} = \lceil B_w D_1 / M \rceil. \quad (2)$$

In addition, as for large size of packets, the Packet-in message can easily exceed the size of Maximum Transmission Unit(MTU) when encapsulates the whole packet into it. Thus, the switch has to split them, which adds extra work to the weak CPU in the switch control plane.

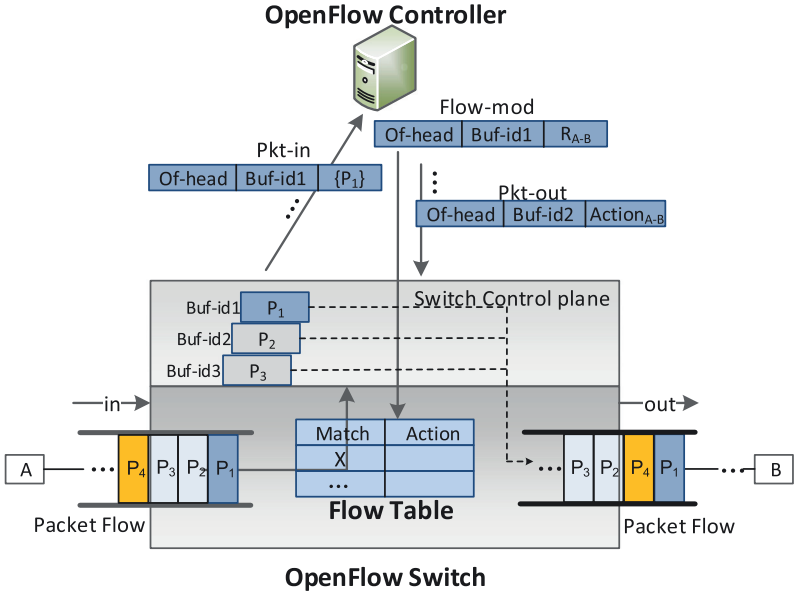
Then we describe the procedure of flow setup with the packet-granularity Packet-in buffer (we call it generalbuffer below) in the existing switch. Responding the message of OFPT_FEATURES_REQUEST, the switch notifies the controller the maximum number of packets it can buffer. After that, the controller sends an OFPT_SET_CONFIG message to configure the maximum size of bytes of the original packet to be carried in the Packet-in (We name those bytes as the digest of the packet). As described in OpenFlow Specification, it is 128 bytes by default. The flow setup with buffered switch is shown in Fig. 2(b). In this case, mismatched packets (P1, P2, P3) are buffered in the switch with Buffer_id1, Buffer_id2 and Buffer_id3 respectively. So the switch generates Packet-in only with the digest and Buffer_id (32 bits). The controller acts almost the same as previous. But the difference is that Buffer_id, not the whole packet, is included in all OpenFlow messages. The switch receives the Flow-mod message including Buffer_id1. In order to reduce the number of mismatched subsequent packets for the flow, it firstly installs the rule R_{A-B} to the flow table, and then release the packet stored in the buffer with Buffer_id1. Other Packet-out messages are processed sequentially by taking out the packet from corresponding Buffer_id and executing actions. As the ratio of packet size between a Packet-in message with Buffer_id and an MTU packet is M_{pi}: M (in Table 1), the load comparison between these two kinds of switches is M_{pi}: M. However, the request number of Packet-in messages to the controller is the same, which is as follows:

$$R_{generalbuffer} = R_{nobuffer} = \lceil B_w D_1 / M \rceil. \quad (3)$$

Although the buffered switch can significantly reduce the load on the OpenFlow channel, it still exists the out-of-order problem and can be further improved. Still considering the example in Fig. 2(b), packets of a new flow arrive the switch as the sequence of P1-P2-P3-P4. P1 is used to generate Packet-in to the controller for the rule. P2 and P3 are also sent due to the delay of the rule. Assuming the rule is installed before P4 looks-up the flow table, so P4 hits the rule and be forwarded. But at the same time, the Packet-out messages for P2 and P3 have not arrived or executed by the control plane of the switch. Therefore, a likely output sequence of the new flow is P1-P4-P2-P3. The problem of out-of-order packets also happens in the OpenFlow switch with no buffer, as shown in Fig. 2(a). Blanton et al. [16] points out that the reassemble caused by out-of-order packets of TCP flows accounts for large buffers at host and decreases the throughput of connections. Although out-of-order packets have less negative effect on connectionless UDP flows, applications in end hosts still need to waste CPU cycles in reassembling out-of-order packets, which slows down the performance of communication.



(a) OpenFlow switch without buffer



(b) OpenFlow switch with packet-granularity Packet-in buffer. {P1} is the digest of P1.

Fig. 2. Comparison of working pattern between different switch designs.

2.3. Our motivation

Mismatched Buffers are needed not only for UDP flows, but also for TCP flows in OpenFlow networks. It is well-known that TCP flows starts up the connection with three-way handshake before transmitting data packets. The SYN and SYN/ACK segments invoke the controller to install rules along the path on all the concerned switches. As a result, all data packets of TCP flows forward based on installed rules. However, it may not be the case in datacenter networks. Benson et al. measured that new flows can arrive at a given switch within 10 μs of each other, which means the switch sees 100 thousand flows per second [15]. Thus the number of active flows existing instantaneously is larger than the limited number of entries of flow table in switch hardware. Besides, Benson also found flows in datacenters have characteristics of ON/OFF. As a result, the rules in switches for already connected TCP flows may easily be replaced in the OFF time of flows.

As for the example in Fig. 2, end host A and B have built TCP connection with each other and the forwarding rules has also been installed. But the rules R_{A-B} in the switch may be substituted due to limited rules' space. Then a burst of TCP data packets in the connection arrives, these packets fail to hit the flow table and are sent to the controller, which is the same as connectionless UDP flows. Therefore, buffers are also useful to prevent TCP flows from overloading the switches and the controller in the environment of datacenter.

As described above, buffers in switches mitigate the communication load between controllers and switches during the flow

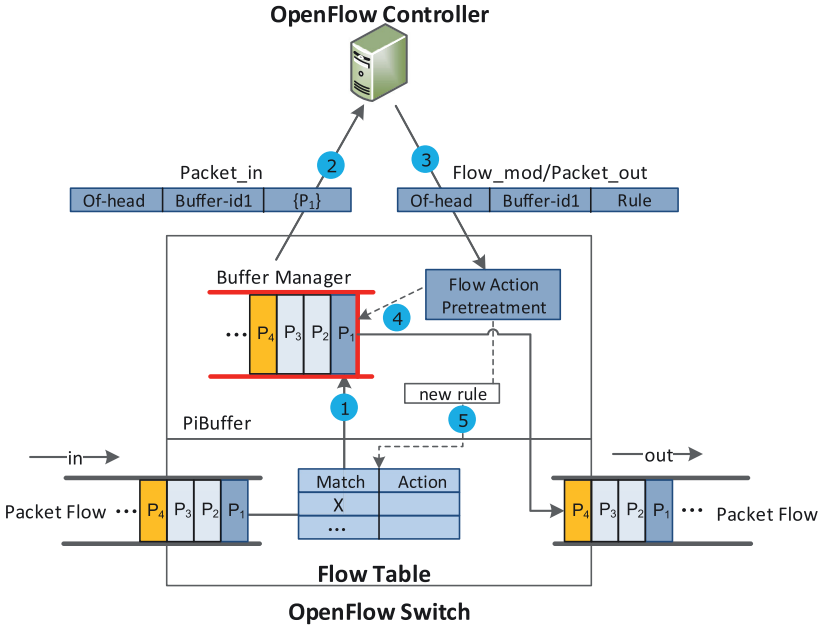


Fig. 3. Flow setup with FPB-based OpenFlow switch.

setup. But it increases the complexity of switch design and the challenge in buffer management. Currently, for considering the cost and simplicity, many commodity off-the-shelf OpenFlow switches adopt nobuffer design. However, Lu et al. proposed to enhance the capability of commodity hardware switch by adding the combination of CPU and DRAM to the switch control plane [21]. Furthermore, many deployed OpenFlow-based datacenters overlay onto the physical network by leveraging software virtual switch which resides in the server as the first-hop switch (e.g., Open vSwitch [22]). It is relatively easy to add a new buffer module to the software switch and the buffer space is sufficient inside the server. As a result, we think it is feasible to introduce a well-managed buffer to OpenFlow switches in SDN deployment.

3. The proposed FPB scheme

We consider an SDN datacenter network where a centralized SDN controller computes the forwarding table and installs rules satisfying the inequation (1).

In order to overcome the defects of buffer management in existing OpenFlow switches, we propose a novel model of flow buffer management under the traffic environment of datacenters, named FPB(short for Flow-granularity Packet-in Buffer). As there is a latency between the switch generating the Packet-in message for a mismatched packet and installing a rule for the packet, some subsequent packets would also get mismatched and be sent to the controller too. With FPB, the OpenFlow switch only sends the first packet of a flow, while other mismatched packets in the same flow are buffered in flow-granularity. Thus it can further reduce the communication load between the control plane and data plane. Besides, FPB leverages flow actions pretreatment to solve the problem of packet out-ordering in a flow, which is caused by the sequence of packet processing with the model of current buffers.

Fig. 3 illustrates the sequence of main operations with the FPB-based OpenFlow switch. We add two components to the control plane of switches: Buffer Management and Flow Action Pretreatment. FPB does not change the OpenFlow pipeline, which achieves the functions of parsing packet headers, looking up flow tables and executing flow actions. It modifies the processing of mismatched packets and the installation of flow rules. Thus, FPB only redefines the processing of three kinds of OpenFlow messages in switches (namely Packet-in messages, Flow-mod messages containing Buffer_id, Packet-out messages containing Buffer_id), while leaves the processing of other OpenFlow messages remaining the same. The workflow of FPB is described as follows.

Table 2
Comparisons of nobuffer, generalbuffer and FPB.

	Nobuffer	Generalbuffer	FPB
Workload usage	$B_w D_1$	$B_w D_1 M_{pi}/M$	1 Packet-in message
Number of Packet-in	$B_w D_1/M$	$B_w D_1/M$	1
Out-of-order	Yes	Yes	No

Step 1: When a new flow arrives at the switch, as no matched rule exists in the flow table, the first packet of the flow and some subsequent packets are buffered in flow-granularity. So Buffer Management only allocates one Buffer_id and one buffer to those buffered packets. The buffer, which is a First-In-First-Out(FIFO) for each flow, stores subsequent mismatched packets by orders.

Step 2: Then the control plane of switch generates one Packet-in message to the controller carrying the Buffer_id and the digest of P1.

Step 3: The controller responds it with a Flow-mod message including the Buffer_id and the rule for the flow. The Flow Action Pretreatment receives the Flow-mod message.

Step 4: It takes out all packets in the buffer with Buffer_id1 and releases them to the OpenFlow pipeline sequentially.

Step 5: Flow Action Pretreatment installs the new rule to the flow table. As a result, the output sequence of the new flow is P1-P2-P3-P4, which is the same as input. In the above processing of the FPB model, a new flow's setup just generates only one Packet-in message and one Flow-mod message to the OpenFlow channel. The comparisons of above three different switch designs (nobuffer, generalbuffer and FPB) are shown in Table 2.

The row "Workload usage" of the table indicates the workload on OpenFlow channel in the duration of $D1$, which is generated by mismatched packets of a new flow. As Packet-in and Packet-out come in pairs, the workload only consists of Packet-in messages

here, and encapsulation of OpenFlow message is ignored. According to the comparisons in Table 2, FPB can significantly reduce the workload caused by burst mismatched packets and keep packets sequential in OpenFlow networks.

Besides, there are two problems that should be considered in the design of FPB. The first one is the buffer policy for FPB. General buffer stores each mismatched packet and has no policy for buffer. But as the buffer resource is limited and expansive, the buffer policy is necessary. Benson et al. [15] measured the distribution of packet sizes, which exhibits a bimodal pattern. Most packet sizes are around either 200 Bytes or 1400 Bytes, while 200-Bytes packets are keep-alive messages of applications or TCP acknowledgments, and 1400-Bytes packets are critical data packets. We may develop a policy based on the size of packets, as small packets are individual usually and have no need to occupy the buffer resource.

The second concern is the capability of switch of handling the buffered packets. FPB realizes installing rules after releasing buffered packets, which creates a great challenge to the processing speed of switch. Assuming the processing speed for releasing the buffer is P_s, the total time for releasing is t_s. If the complete time of the flow t_{flow} is larger than D_1, namely $t_{flow} > d_{S1\to C} + d_C + d_{C\to S1}$, the processing speed should satisfy the following inequation:

$$P_s \geq \frac{B_w(D_1 + t_s)}{t_s}, (if, t_{flow} > D_1), \tag{4}$$

If the processing speed of releasing the buffer is not fast enough, it will cause mismatched packet buffer overflow and serious packet loss. Therefore, Eq. (4) specifies the requirement for the processing speed of releasing the buffer, which is the capability of switch to handle the buffered packets. Otherwise, it may cause mismatched packet buffer overflow and serious packet loss.

Since FPB protects the controller from being overloaded by burst mismatched packets, the workload of controller is relatively less compared to previous models. Therefore the controller processes Packet-in message and installs rules with less response time, which decreases the completion time for each reactive flow. The cost of FPB is the buffer resource and complicated buffer design. Although buffer size is not of concern in software implementation, commodity hardware switch's buffer size is usually small in order to reduce cost. We denote the *Profit* as the benefit by introducing buffer resources to OpenFlow switches, as shown in Eq. (5). There are m flows coexisting in the network. The reduction of flow completion time is tc_{flow}, while the buffer size for flow is $buffersize_{flow}$. Therefore *Profit* can be quantified as:

$$\text{Profit} \equiv \sum_{i=1}^{m} tc_{flow_i} / \sum_{i=1}^{m} buffersize_{flow_i}. \tag{5}$$

4. Detailed design of FPB

We now present the detailed design of FPB, which is an enhancement to the control planes of OpenFlow switches. Based on the basic OpenFlow pipeline, we redefine the buffer operation for mismatched packets and the processing of OpenFlow messages containing Buffer_id, which are described in Sections 4.2 and 4.3.

Fig. 4 shows a conceptual diagram of the overview of our proposed FPB. It mainly consists of two modules: Buffer Management and Flow Action Pretreatment. Due to space limitation, other common OpenFlow related modules (e.g. OpenFlow messages encapsulation/decapsulation) in switch control planes are omitted in the figure.

Buffer Manager. This module buffers mismatched packets in flow-granularity. It contains four components: buffer creator, buffer releaser, Packet-in Buffer Table(PiBT) and buffer area. PiBT is the core component of our proposed model, which would be described later. The buffer creator is responsible for deciding whether to store the first mismatched packet based on the buffer policy and space. If such condition is met, it inserts a new entry to PiBT for the new flow. Then it generates a Packet-in message with allocated Buffer_id to the controller. The buffer releaser takes the Buffer_id and flow actions as inputs, takes out packets from buffer area according to the Buffer_id and sends those packets to OpenFlow pipeline.

Flow Action Pretreatment. This module is responsible for the pretreatment of OpenFlow messages carrying Buffer_id. It contains two components: action parser and rule installer. The action parser only processes the Flow-mod and Packet-out messages with Buffer_id, and then it informs Buffer Manager releasing the packets according to the Buffer_id. The rule installer inserts flow rules to flow table. When processing Flow-mod messages, the module enforces switch to install rules to flow table only after receiving the processing state of buffer empty for the Buffer_id.

4.1. Packet-in Buffer Table

Each entry of PiBT records the buffer information of a flow, which includes the match field, start index, current index, packet count and timeout, as shown in Fig. 5.

The match field is the identifier of a flow, namely *flow_key*. It is a subset of the packet digest and is defined according to different demands. For example, the following five tuple is commonly used, which consists of IP source address, IP destination address, IP protocol type, TCP/UDP source port and TCP/UDP destination port. Each mismatched packet looks up the PiBT by comparing the *flow_key*. According to the result, the PiBT decides to whether create a new entry or add the packet to the corresponding entry's buffer area. The start index and current index are used to indicate the places where the first packet and the latest packet from a mismatched flow store respectively. The start index points to a packet and current index is NULL when the entry is created and initialized, while the start index equals to current index if the entry's buffer area is empty, which means that the buffered packets have already been released. The index represents different meanings for different implementations. It means the hardware address when the buffer is realized in hardware, while it means the pointer of the buffer when realized in software. The packet count records the number of packets stored in the buffer for a flow. The timeout is the live time of the entry. If FPB does not get the respond message within the timeout for a Packet-in message about a buffered flow, it will delete the entry, set free the buffer space of the flow and drop all packets of the flow.

The buffer area stores the packets in a flow as the singly-linked list. When storing a packet of an existing entry, it adds the packet to the next index of currently last packet of the flow and updates the current index of the entry. If the buffer is realized in software, the size of buffer space is allocated dynamically according to the size of buffered packet. However, it is quite difficult to allocate the right buffer space for the packet in hardware. Therefore, allocating each packet with a fixed buffer size in advance, such as the size of MTU, may be applied in hardware. As mentioned before, the distribution of packet size in the datacenter network presents a bimodal pattern. We can make a buffer policy based on the size of packets, which only stores large packets. So even the buffer space is allocated earlier, it is not wasted and used sufficiently.

The number of PiBT entries indicates the number of mismatched flows that the switch can maintain. So more entries means that the switch has the capability to adapt to more dynamic

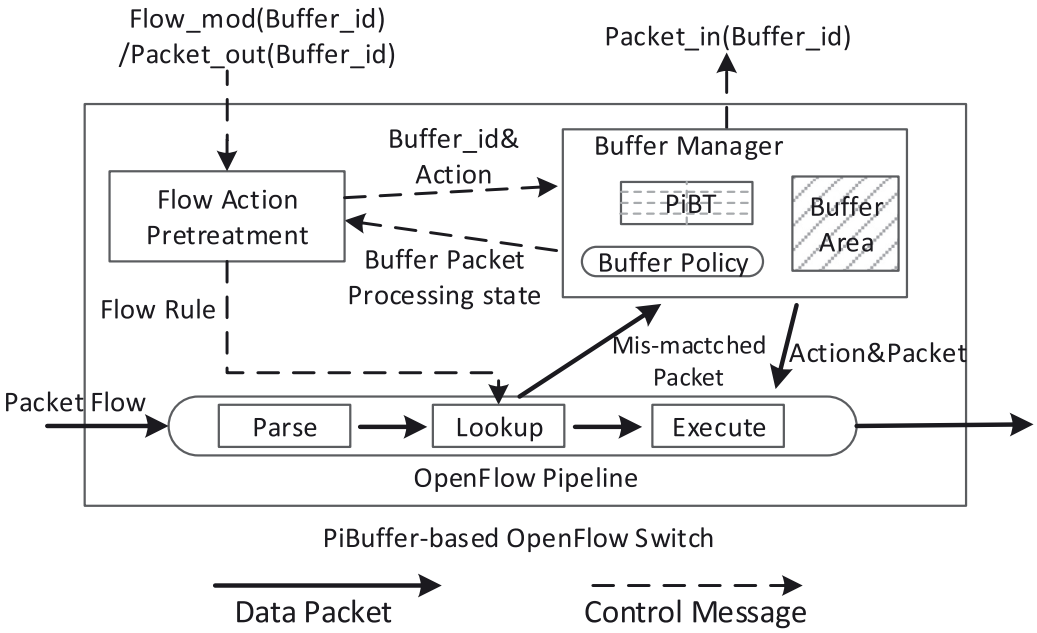


Fig. 4. The structure of FPB.

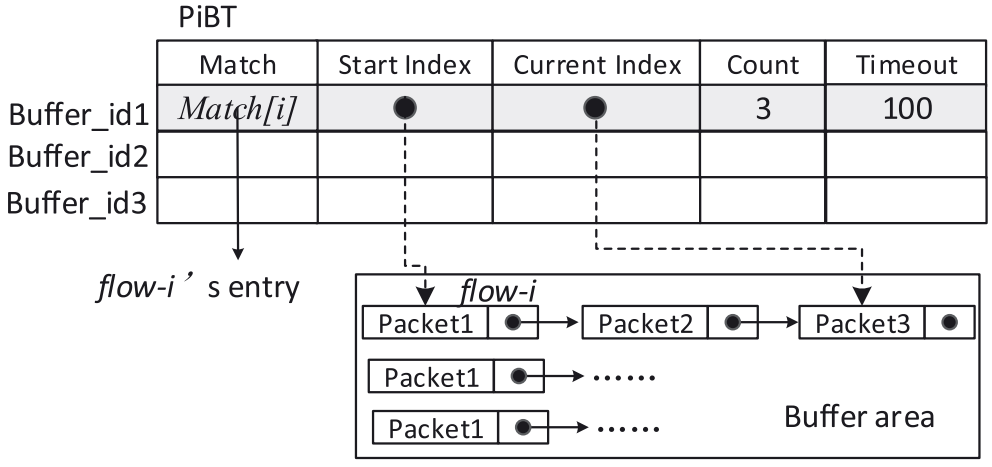


Fig. 5. The structure of Packet-in Buffer Table.

flows in the network. But the number is limited by the buffer resource of switches.

4.2. Buffering mismatched packets in flow-granularity

When FPB receives a mismatched packet from the OpenFlow pipeline, Buffer Manager firstly decides to whether store the packet according to the buffer policy and buffer usage. The packet is sent to the controller as a Packet-in message with the whole packet when it is conflicting with the buffer policy or the buffer is full. Otherwise, Buffer Manager uses the *flow_key* of the packet, which comes from OpenFlow pipeline, to look up PiBT. If present, it adds the packet to the next point of current packet in the buffer area by access the current index of the matched entry, and updates the current index and count of the entry. Then the packet is buffered without generating Packet-in or getting loss. If the matching fails, Buffer Manager allocates a new buffer space, and inserts a new entry to PiBT incrementally with the *flow_key*. It returns the index of PiBT as the Buffer_id of the packet. At last, the switch control plane generates a Packet-in message with the Buffer_id and digest of the packet. The basic algorithm is described in pseudo-code in Algorithm 1.

Algorithm 1 Buffering mismatched packets in flow-granularity.

Require: Mismatched Packet with *flow_key*.
Ensure: *Packet-in* Message.

```
1: if (Mismatched packet conflicts with buffer policy or buffer is full ) then
2:     buffer_id = NULL;
3:     Generate Packet-in message with NULL buffer_id;
4: else if (Exist an flow entry in PiBT) then
5:     Update PiBT with current packet index;
6: else
7:     Allocate a buffer for the packet;
8:     Generate a buffer_id for the packet;
9:     Generate Packet-in message with buffer_id;
10: end if
```

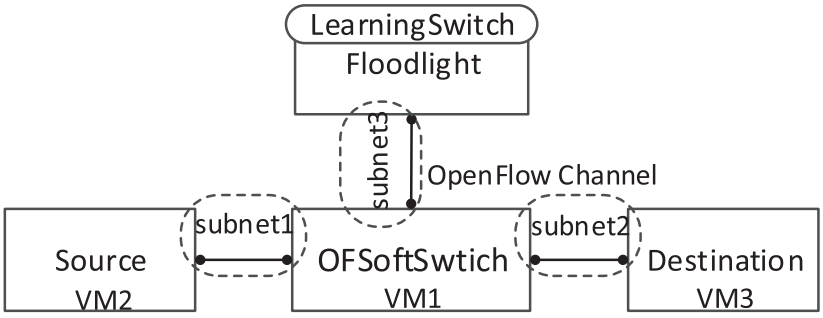


Fig. 6. Experimental topology with OFSoftSwtich.

4.3. Flow Action Pretreatment

Here we present how FPB interacts inside the model when the switch receives the Flow-mod /Packet-out message carrying Buffer_id from the controller. The action parser component in Flow Action Pretreatment extracts the Buffer_id and flow action from the OpenFlow message, and sends them to Buffer Manager. After receiving the Buffer_id and flow action, the buffer releaser component of Buffer Manager finds the entry in PiBT based on the Buffer_id. According to the field of start index in the entry, the buffer releaser sequentially takes out the packets from the buffer area and sends each packet along with its action to the OpenFlow pipeline one by one. When all packets are released, it deletes the entry with the Buffer_id and sends a complete signal of buffer packet processing state to Flow Action Pretreatment. If the OpenFlow message is Flow-mod, the rule installer of Flow Action Pretreatment then installs the rule to the flow table. The procedure is described in pseudo-code in Algorithm 2.

Algorithm 2 Processing OpenFlow message with Buffer_id.

Require: OpenFlow message *OFMessage*.
Ensure: Buffered packet, flow action and forwarding rule.

```
Extract buffer_id from OFMessage;
if (PiBT exists flow entry with buffer_id) then
    Extract actions from OFMessage;
    while (PiBT exists packets with buffer_id) do
        Extract packets from the buffer;
    end while
end if
if (OFMessage is Flow-mod message) then
    Install forwarding rules to flow table;
end if
```

5. Performance evaluation

In the deployment of OpenFlow networks, the OpenFlow switch is realized in software as well as in hardware. Thus, we validate our proposed design of buffer both in software and hardware in this paper.

5.1. Evaluation in software prototype

OFSoftSwtich is an open source software switch implemented in user mode [12]. So it is easy to modify and develop to validate new functions. We realize the FPB as a module in C and add it to the software switch. Our target is to compare the performance of FPB with the existing generalbuffer and nobuffer.

All experiments are conducted in a computer with Intel Core i5 3.2 GHZ of CPU and 8 GB of memory. We build a simple topology in the host, as shown in Fig. 6. Since our paper is founded on the assumption of inequation (1), we only consider the buffer of mismatched packets at the first-hop switch. So this simple scene is enough for evaluating the buffer. The controller uses modular Floodlight and runs an application named LearningSwitch [23] in it, which installs rules for MAC based forwarding. Three virtual machines noted as VM1, VM2 and VM3 are run in VMware. VM2 and VM3 are regarded as the source and the destination of flows respectively. VM1 runs an instance of OFSoftSwtich and has three virtual network interface cards (vNIC), which connect to Floodlight, VM2 and VM3 respectively. All three connections are in different subnets. Thus only rules installed to the OFSoftSwtich by the controller can VM2 send flows to VM3.

We first validate that FPB can reduce communication costs between the controller and OFSoftSwtich. As explained in Section 2.3, like UDP flows, a large number of data packets in TCP flows may mismatch the flow table under the condition of traffic characteristic in datacenter networks. Therefore, we only use UDP flows in this test. Iperf is used for generating flows, where VM2 is set to the UDP client and VM3 is set to the UDP server. The bandwidth of flows is 5 Mbps and the flows all last 60 s in each test. For comparison, the instance of OFSoftSwtich is configured to nobuffer, Generalbuffer and FPB. In order to present the bandwidth occupied by OpenFlow messages in the flow setup, the controller does not install rules in tests. So all packets will fail to hit the flow table and be processed by the switch and the controller according to different configurations. The result is shown in Fig. 7.

The bandwidths of “Send” and “Receive” are described in the point of view of the switch, which correspond to Packet-in messages and Packet-out messages respectively. Obviously, the FPB-based switch takes the least bandwidth of OpenFlow channel, which averages 74 KB/s (Send) and 23 KB/s (Receive). The average utilized bandwidth for Generalbuffer is 139 KB /s (Send) and 43 KB/s (Receive). As for nobuffer, the average is 761 KB /s (Send) and 719 KB/s (Receive), which almost equals to the bandwidth of the flow sent from VM2 to VM3 (5 Mbps). The utilized bandwidth for nobuffer comparing to Generalbuffer is about the ratio of 1:5, as the length of is M_{pi} is about 300 Byte, while M is 1500 Byte. When a switch has a buffer, it sends the Packet-out message only containing a 32-bits Buffer_id, which is one third of the Packet-in message. It also satisfies the bandwidth ratio of Send and Receive with the buffered-switch observed from the result. Thus FPB can significantly decrease the bandwidth usage of OpenFlow channel.

Then we present the benefit of bandwidth reduction to the controller and switch, as shown in Fig. 8. The x-axis shows the CPU utilization of VM1 running OFSoftSwtich by percentage, and the y-axis shows the result of CDF. When encountering burst of a flow, instead of sending each mismatched packet (or digest of the packet) to the controller, FPB only sends the first one and stores subsequent packets in flow-granularity. Since it is more costly of CPU to generate and send Packet-in messages than store packets in the switch, FPB takes about 25% less CPU resources than Gen-

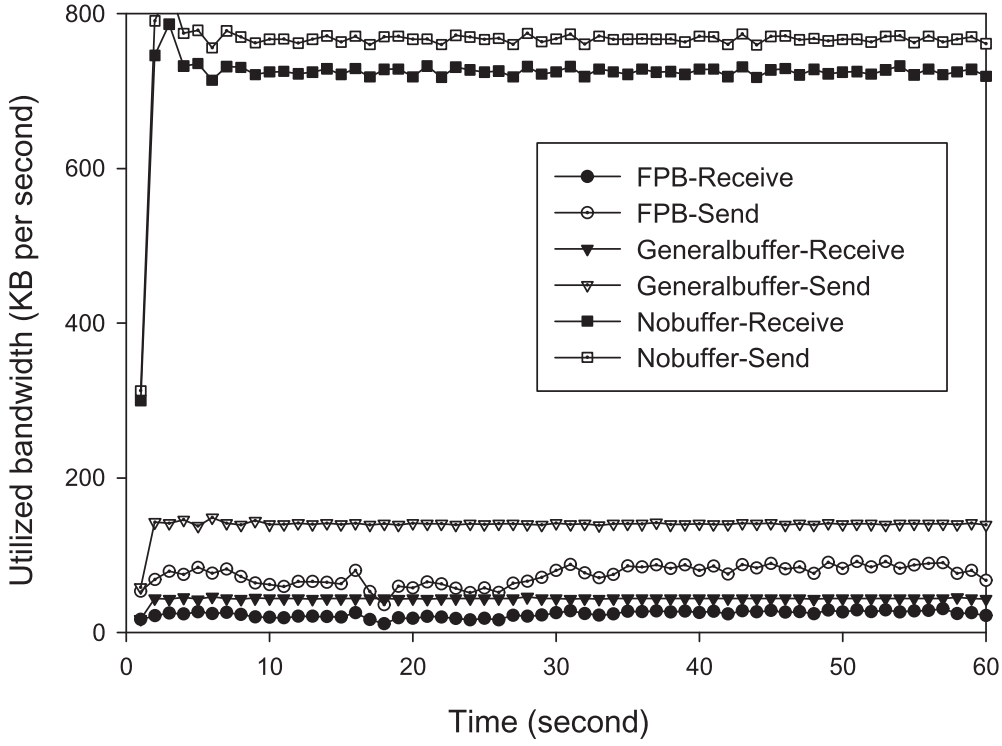


Fig. 7. The bandwidth utilization in 60 s at the VM running OFSoftSwtich with different buffer designs.

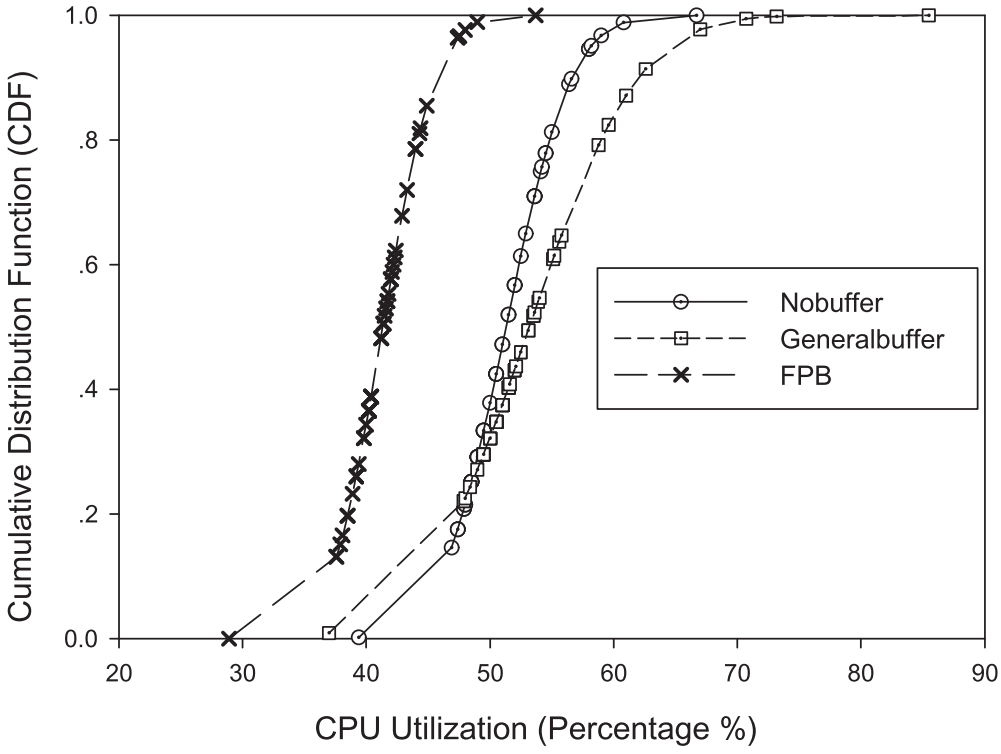


Fig. 8. A CDF of the CPU utilization in the VM running OFSoftSwtich with different buffer designs.

eralbuffer. Can be expected that the controller consume fewer CPU cycles for one flow setup due to fewer Packet-in messages to the controller. It also reduce the possibility of inconsistency of control logic as only one packet of a flow is sent to the controller. The CPU utilizations of nobuffer and generalbuffer are almost the same as the request number of both models is equal, which validates our analysis in Eq. (3).

In addition, we also validate that FPB can solve the problem of out-of-order packets in a flow. In the test, VM2 sends a sequence of packets continuously without no pauses. Each packet carries a sequence ID, which starts from 1 to 100. The sequence ID received at VM3 is recorded in Fig. 9. For Generalbuffer, the result reflects the same appearance of out-of-order packets as analyzed in Section 2.2. VM3 receives ID 1 first, and then ID 64 along to 97,

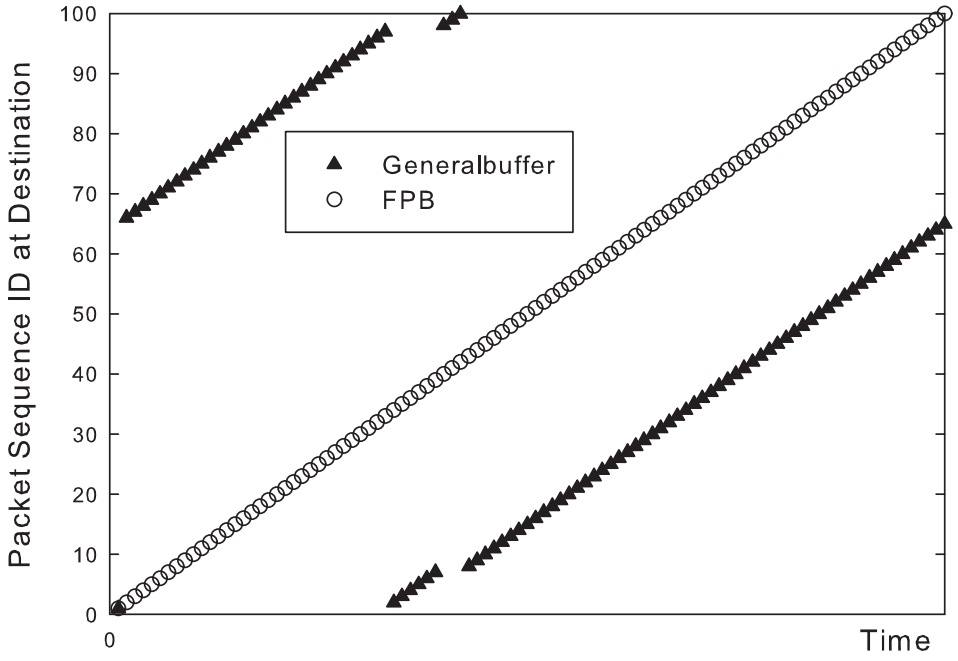


Fig. 9. The packet sequence number over time received at VM3.

Table 3
Resource usage in FPGA.

Resource	Total	All modules usage(%)	FPB related usage(%)
ALUTs	36,100	13,528(37.5%)	627(1.7%)
Registers	36,100	15,620(43.3%)	552(1.5%)
Memory bits	2,939,904	1,728,640(58.8%)	74,496(2.5%)

then ID 2. The out-of-order packets can decrease the bandwidth of TCP flows extremely. But for FPB, VM3 receives those 100 packets sequentially from 1 to 100.

To sum up, when the buffer resource is sufficient in OpenFlow switches, well-managed buffers can not only prevent large number of mismatched packets from overloading switches and controllers, thereby decreasing the CPU utilization of both switches and controllers, but also realize order-preserving transmission for inner packets of a flow.

5.2. Evaluation in hardware prototype

On the other way, we evaluate how buffer changes over time by realizing the proposed model of buffer management in hardware programmable NetMagic platform [17]. Like the NetFPGA, to ease the complexity of development, the NetMagic platform provides a simple and rapid development model, clear module interfaces for hiding basic common packet processing logic in hardware, and several standard reference designs including a simplified OpenFlow switch [24]. It allows users to focus on the implementation of self-defined functional modules. We add FPB-related hardware implementation to the existing simplified OpenFlow switch on NetMagic. The resource usage of FPGA is presented in Table 3. Although simplified, it is feasible to realize proposed buffer functions in hardware. In the implementation, for simple calculation, every mismatched packet is allocated with the size of 2 KB memory in chip.

Fig. 10 shows the deployment and steps of the test. There are three computers, one is Controller and the other two are Source and Destination respectively. Besides, the NetMagic switch connects these three computers. At start up, there is no rule in the NetMagic switch. Thus, as Source sends sequential packets to the Destination, all packets are buffered in the controller until the controller installs the rule. In the controller, we vary the delay between sending the rule and receiving the first Packet-in message from the switch, namely d_C in Table 1. The delay increases progressively with 50 ms from 50 ms to 500 ms. After installing the rule, the controller collects the number of packet count in PiBT. The experiment of every delay runs 10 times. Results are shown in Fig. 11.

We can see that the size of buffer jitters more severely as the delay increases. It is because that the delay of installing the rule inside a switch is easily affected both by processing load of the switch and controller. Longer delay incurs more unstable states in the switch. Thus, the delay happened in the rule installation has negative effect on the size of buffer and it should be shortened as long as possible.

Since storage is relative sufficient in software, buffer resource is not basically restricted for FPB-enabled OpenFlow software switch. However, in hardware switch, the buffer size is a critical factor in the design of OpenFlow hardware switch. In the experimental results, we can observe that estimating the total required buffer size of switch will be critical for OpenFlow-based hardware switches. The total required buffer size can be estimated by analyzing the traffic flows and the transmission delay (response time) from the controller, which may be significantly affected by real network traffic (traffic flow, traffic model etc.), controller deployment and controller processing capability, etc. For example, a high-performance Openflow controller with specified source-destination traffics will require lower buffer storage than a low-performance controller with fluctuating traffics in the switch. However, estimating the total required buffer size in Openflow switches is out of the scope of this paper. We will consider this interesting issue as a future work.

6. Related work

As the network scales out, it is necessary to avoid the centralized controller become the bottleneck of the performance of network. This problem is emergent and critical when deploying SDN

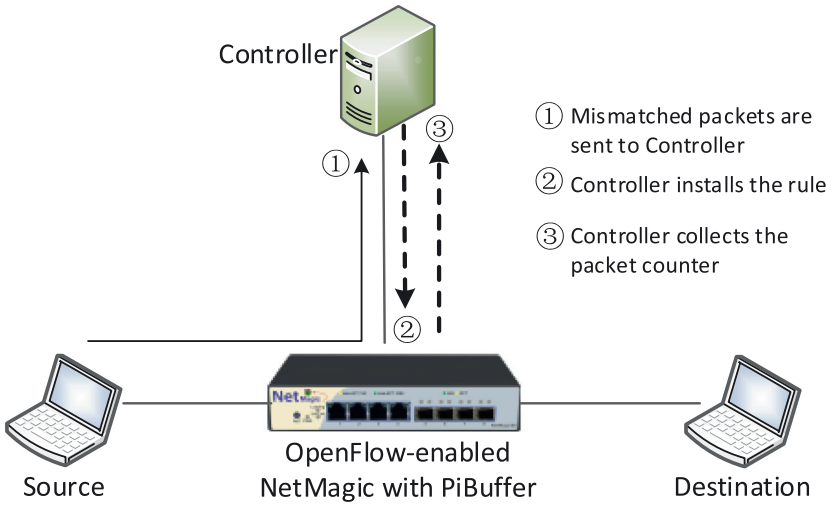


Fig. 10. Experimental topology with NetMagic.

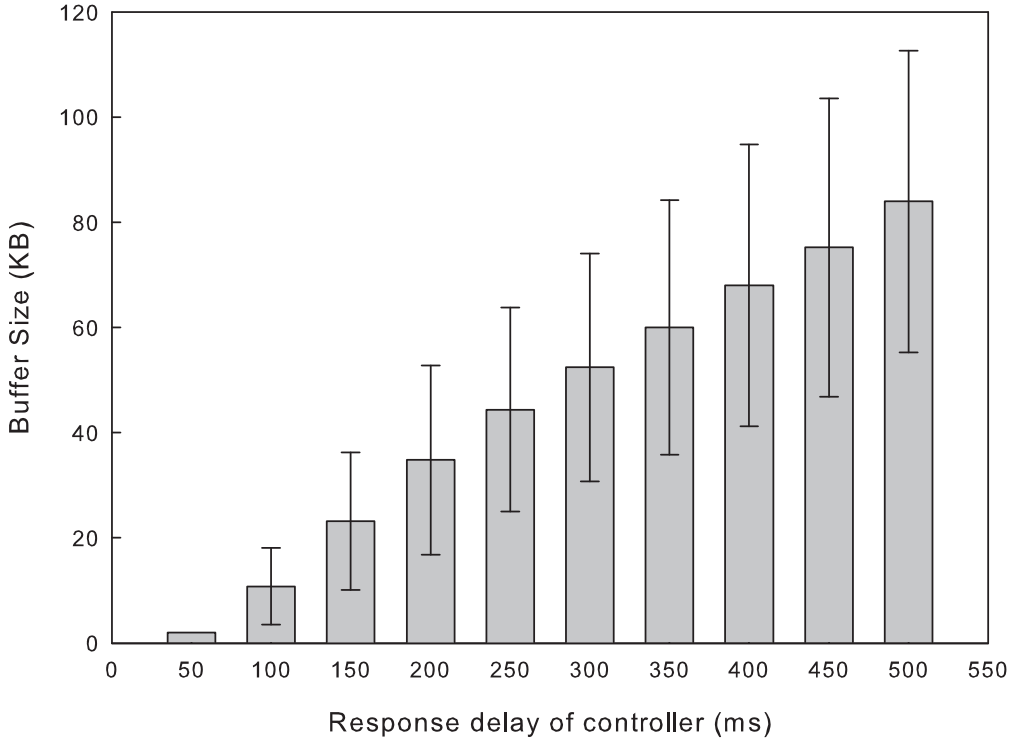


Fig. 11. The controller response delay varies over buffer size in NetMagic.

in datacenter networks. There are two directions to mitigate the overload of single-point centralized controller. One is to improve the processing capability of the SDN control plane. The other is to reduce the communication workload on OpenFlow channel as much as possible by enhancing the function of the SDN data plane.

In the first aspect, the developer of SDN controllers adopts the techniques of multi-thread, shared queue, batch process and so on, to design a high-performance controller. As the single-point controller has the upper limit in performance, researchers have designed a distributed architecture of SDN control plane. HyperFlow [7] deploys multiple distributed controllers to share the overhead between the control plane and data plane. Each controller in Hyperflow synchronizes the network-wide view by using publish/subscribe messaging paradigm. Onix [8] also distributes the network view among multiple controller instances, but it uses the network information base to maintain the consistency of the underlying network state. ONOS [9] is an open source distributed SDN control platform, which gets more attentions recently. All these works are complementary to our work.

In the other aspect of data plane enhancement, the most related work with ours is the packet-in message filtering mechanism proposed by Kotani et al. [10]. We both aim at protecting the OpenFlow control channel from too many mismatched packets that bring high loads. Similar to FPB, the filtering mechanism also records the values of packet header fields before sending Packet-in messages. However, the difference is that it filters out the packets that have the same values as the recorded ones, while we buffer those packets in flow-granularity, which causes less packet loss. In addition, DIFANE [11] proposes to pre-distribute rules to authority switches. Consequently, for better performance and scalability,

it keeps all traffic in the data plane without directing packets to the controller. Devoflow uses rule clone and local actions for devolving control to the switch. It also reduces the need to transfer statistics for boring flows by achieving efficient statistics collection in the switch. Our method is also a means of the SDN data plane enhancement by leveraging the buffers in OpenFlow switches.

As for the problem of out-of-order packets during the flow setup, previous work have tried to solve the problem through sophisticated controlling in the controller [25]. However, we think the root cause for the packet out-ordering is involving in the switch. So we design the mechanism of flow action pre-processing to realize order-preserving forwarding in switches.

7. Conclusions and future work

In this paper, we exploit the buffer resource in switches and propose FPB, a flow-granularity buffer for mismatched packet. FPB records the flow information of the first packet before sending Packet-in messages and buffers subsequent packets in flow-granularity. It guarantees the order of packets in the flow by releasing the buffer sequentially. As we expected, our experiment results show that switches with FPB dramatically reduce the number of switch-controller interactions, which mitigates the load of controller. FPB also decreases CPU utilization in the switch control plane. We believe that FPB can be regarded as a reference in the design of hardware or software OpenFlow switches to improve the scalable of OpenFlow networks.

In the future work, we will estimate the impact of flow traffic and controller response time on the required total buffer size.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable feedback. This work was supported in part by grants from Startup Research Grant of National University of Defense Technology (No.JC15-06-01), Chinese National Programs for High Technology Research and Development (863 Programs) (No.2013AA013505) and Youth Natural Science Foundation of China (No.61601483).

References

[1] M. Al-fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat, Hedera: dynamic flow scheduling for data center networks, in: Proceedings of 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2010, p. 19.
[2] A.R. Curtis, J.C. Mogul, J. Tourrilhes, P. Yalagandula, Devoflow: scaling flow management for high-performance networks, in: Proceedings SIGCOMM, 2011, pp. 254–265.
[3] B. Stephens, A. Cox, W. Felter, C. Dixon, J. Carter, Past: scalable ethernet for data centers, in: Proceedings of the ACM International Conference on emerging Networking EXperiment and Technologies (CoNEXT), 2012, pp. 49–60.
[4] T. Koponen, K. Amidon, P. Balland, M. Casado, et al., Network virtualization in multi-tenant datacenters, in: Proceedings of 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2014, pp. 203–214.
[5] M. Banikazemi, D. Olshefski, A. Shaikh, J. Tracey, G. Wang, Meridian: an sdn platform for cloud network services, IEEE Commun. Mag. 51 (2) (2013) 120–127.
[6] Openflow specification 1.4, Open Networking Foundation http://www.opennetworking.org (2013).
[7] A. Tootoonchian, Y. Ganjali, Hyperflow: a distributed control plane for open flow, in: Proceedings of the Internet Network Management Workshop on Research on Enterprise Networking (INM/WREN), 2010, p. 3.
[8] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, S. Shenker, Onix: a distributed control platform for large-scale production networks, in: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI), 2010, pp. 1–12.
[9] P. Berde, M. Gerola, J. Hart, et al., Onos: towards an open, distributed sdn os, in: Proceedings of the SIGCOMM Workshop on Hot Topics in SDN, 2014, pp. 1–6.
[10] D. Kotani, Y. Okabe, A packet-in message filtering mechanism for protection of control plane in openflow networks, in: Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), 2014, pp. 29–40.
[11] M. Yu, J. Rexford, M. Freedman, J. Wang, Scalable flow-based networking with difane, in: Proceedings of the SIGCOMM, 2010, pp. 351–362.
[12] Ofsoftswtich, http://cpqd.github.com/ofsoftswitch13.
[13] Open vswitch: an open virtual switch, http://www.openvswitch.org (2014).
[14] K. Phemius, M. Bouet, Openflow: why latency does matter, in: Proceedings of the Internet Network Management Workshop on Research on Enterprise Networking (INM/WREN), 2013, p. 3.
[15] T. Benson, A. Akella, D. Maltz, Network traffic characteristics of data centers in the wild, in: Proceedings of the ACM/USENIX Internet Measurement Conference (IMC), 2010, pp. 267–280.
[16] E. Blanton, M. Allman, On making tcp more robust to packet reordering, in: ACM Computer Communication Review, 2002, pp. 20–30.
[17] T. Li, Z. Sun, C. Jia, Q. Su, M. Lee, Using netmagic to observe fine-grained per-flow latency measurements, in: Proceedings of the SIGCOMM Demo, 2011, pp. 466–467.
[18] M. Fernandez, Comparing openflow controller paradigms scalability: reactive and proactive, in: Proceedings of 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), 2013, pp. 1009–1016.
[19] K. Agarwal, C. Dixon, E. Davis, J. Carter, Shadowmacs: scalable label-switching for commodity ethernet, in: Proceedings of the SIGCOMM Workshop on Hot Topics in SDN, 2014, pp. 157–162.
[20] M. Canini, D. Venzano, P. Peresini, D. Kostic, J. Rexford, A nice way to test openflow applications, in: Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2012, p. 10.
[21] G. Lu, C. Guo, Y. Li, Z. Zhou, T. Yuan, H. Wu, Y. Xiong, R. Gao, Y. Zhang, Serverswitch: a programmable and high performance platform for data center networks, in: Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2011, p. 2.
[22] 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), 2015, pp. 117–128.
[23] Floodlight, http://floodlight.openflowhub.org/.
[24] Netmagic, http://www.netmagic.org/.
[25] P. Peresini, M. Kuzniar, N. Vasic, M. Canini, D. Kostic, Of.cpp: consistent packet processing for openflow, in: Proceedings of the SIGCOMM Workshop on Hot Topics in SDN, 2013, pp. 97–102.

Jianbiao Mao obtained his bachelor's and master's degree in computer science from National University of Defense Technology (NUDT), ChangSha, China. He is currently a Ph.D. candidate at NUDT, under the supervision of Professor Xicheng Lu. His research area is primarily focused on the design and development of Software-Defined Networking (SDN) architectures, datacenter network.

Biao Han is currently an assistant professor at the College of Computer, National University of Defense Technology (NUDT), ChangSha, China. He received the Ph. D. degree in computer science from University of Tsukuba, Japan in 2013. Before that, he received the B.E. degree and finished the master program in computer science in 2007 and 2009 respectively, both from NUDT. His research interests are in computer networks, software-defined networking (SDN), wireless communications and mobile computing. He has been a visiting student in the Department of ECE at University of Florida, USA. He received the IEEE LANMAN Best Paper Award in 2014.

Zhigang Sun is a professor at College of computer of National University of Defense Technology (NUDT), ChangSha, China. He received his Ph.D. in NUDT in 2001. His main research interests include network architecture and design for next generation Internet.

Xicheng Lu is a professor at College of computer of National University of Defense Technology (NUDT), ChangSha, China. His main research interests include computer network architecture.

Ziwen Zhang is an assistant professor in College of Computer, National University of Defense Technology (NUDT), ChangSha, China. He received his B.S., M.S. and Ph.D. degree from NUDT in 2006, 2008 and 2013 in Computer Science. His major research interests include datacenter networks and high performance interconnection.