Queueing Theory over OpenvSwitch: Performance Analysis and Optimization

Fuliang Li[1,2], Naigong Zheng[1], Yuchao Zhang[3], Yishuang Ning[4], and Xingwei Wang[1]

[1] Northeastern University, Shenyang, P.R. China
[2] Beijing National Research Center for Information Science and Technology, Beijing, China
[3] Beijing University of Posts and Telecommunications, Beijing, P.R. China
[4] Kingdee International Software Group Co., Ltd, Shenzhen, China

Abstract. Software Defined Networking (SDN) offers programmability and flexibility by decoupling control plane from data plane. However, its centralized control principle leads to various known performance issues on data plane, e.g., a mismatch packet on data plane will ask control plane how to forward this packet, resulting in extra packet processing delay. In this paper, we illustrate OpenvSwitch (OVS) as an example to investigate performance of SDN switches based on queueing theory. First of all, we describe the architecture and internal workflow of OVS according to its specifications. Then, we present a queueing network model for OVS. The proposed model is able to evaluate the primary influencing factors on performance, including packet arrival rate, table miss probability and packet scheduling policy. In addition, we optimize the established model against these influencing factors. We also model the built-in buffer with queueing theory and reveal how buffer size affects the performance. Experimental results show that both the proposed optimized model and a reasonable buffer size setting can improve the performance of OVS effectively.
Keywords− Software-Defined Networking; queueing theory; performance analysis; OpenvSwitch

1 Introduction

The control plane is decoupled from the data plane in Software Defined Networks (SDN), providing a logically centralized platform to program the state of data plane [1]. SDN switches on data plane are responsible for forwarding data flows according to forwarding rules generated by control plane. Control plane communicates with data plane via OpenFlow or OpenFlow-like protocols. SDN could offer differentiated services for different applications and responds to high availability requirements [2]. However, it also faces many performance issues, e.g., packet processing delay is increased due to frequent interactions between data plane and control plane.

For the incoming packets of a flow, the switch first lookups the flow table that stores the forwarding rules. If there is a forwarding rule for this flow, the packets forward directly according to the rule matching actions. Otherwise, these mismatch packets will be sent to the controller asking for forwarding decisions by default. The controller decides how to forward these packets and sends operation messages back to the switch. Then the mismatch packets and the subsequently arrival packets of this flow are forwarded. According to the above analysis, it can be seen that packet processing delay includes three parts: 1) forwarding operations in the switch; 2) two-way interactions between the controller and the switch; 3) forwarding decision making in the controller. In this paper, we take OpenvSwitch (OVS) as a an example to investigate the performance of SDN from the perspective of model analysis. OVS is a virtual and widely used switch with flexible and programmable ability [3, 4].

Some studies have conducted on the performance of SDN with mathematical methodologies. Azodolmolky *et al.* [5] describe functionalities of SDN with a model, which is built based on network calculus. It is the first time that network calculus is utilized to model the behaviors of SDN. Delay and queue length boundaries, as well as the buffer length are analyzed. Jarschel *et al.* [6] utilize the feedback orientated queueing theory to evaluate the interactions between control plane and data plane. Markovian servers are adopted for SDN, *i.e.* an M/M/1 for the switch and an M/M/1/m for the controller. The main difference between queuing theory and network calculus is that the former is used to model performance of a system under stable state, while the later calculates the boundaries of a system in the worst cases.

Understanding the performance and limitations of SDN is an crucial issue for real deployment. Existing studies have proved the benefits of mathematical analysis models for performance analysis of SDN. However, few studies focus on how to improve the performance based on the analysis models. Different from previous studies, we propose a serious of models for OVS according to its specifications [7]. We not only reveal how the influencing factors affect the performance with a queueing network model, but also evaluate how to improve the performance based on the optimized models. The main contributions of this paper are summarized as follows.

- First of all, we present a queueing network model on the basis of workflow of OVS. We first divide packet processing into several phases and establish queueing network model at each processing phase. We then evaluate the proposed models across different influencing factors, including packet arrival rate, table miss probability and packet scheduling policy.
- To reduce the packet processing delay, we optimize the proposed model from many aspects, including multi-threaded processing, priority queue settings and different packet scheduling policies. We evaluate the optimized models through a comparison analysis.
- To reduce interaction delay between the controller and the switch, we model the built-in buffer based on queueing theory. We interpret how buffer size affects packet processing delay with the built-in buffer model.

The remainder of this paper is organized as follows. Section II presents the related studies. Section III provides an overview of OVS. Section IV introduces the queueing network models. Section V evaluates the established models with experimental analysis. We conclude the paper in Section VI.

2 related work

Considering different influencing factors, existing studies have proposed some approaches to evaluate and improve the performance of SDN. Ansell *et al.* [8] present a network performance prediction tool based on queueing analytic models and couple with real-time measurement. It has the ability of examining how the performance are affected by the changes of traffic load and link utilization. Muhizi *et al.* [9] evaluate the performance of SDN with queuing network models, which could observe the changes of packet processing delay under different parameter settings. Shang *et al.* [10] model packet processing delay of SDN switches and controller. They mainly investigate how *packet_in* messages affect the performance. Wang *et al.* [11] evaluate the throughput and delay of control plane based on queuing theory. How the number of switches, as well as the the number of threads affect throughput and delay are studied. Mahmood *et al.* [12] propose a *Jackson* network, which is used to model data plane, while the controller is modelled as an M/M/1 queue with an infinite buffer or with a finite buffer. Haiyan *et al.* [13] propose a queueing estimation model and extend it for end-to-end delay analysis. Singh *et al.* [14] use queueing theory to model SDN switches from two aspects, *i.e.*, a shared buffer for both control plane traffic and data plane traffic, and a buffer with two priority queues isolating control plane traffic from data plane traffic. Fahmin *et al.* [15] combine SDN with Network Functional Virtualization (NFV) to cope with performance issues. They aim at modelling SDN with NFV with or without the controller. The M/M/1 queuing model is utilized to evaluate the performance.

As a supplement to previous studies, we first model packet processing of OVS to show how the performance is affected by various influencing factors, and more importantly, we verify how to improve the performance with the optimized models against these influencing factors.

3 Overview of OpenvSwitch

This section describes the architecture and working principles of OVS. OVS is a widely used virtual switch for studying OpenFlow networks[3]. It usually works with a centralized controller, which determines the path of a flow by modifying the flow table inside OVS. When packets of a flow cannot match any rules, *packet_in* messages are sent to the controller to request forwarding decisions. On the other hand, OVS can also run without the controller. It forwards the packets according to the messages of layer logic. In this paper, we focus on the former case considering the interaction between OVS and the controller.

As depicted in Fig.1 there are two major components in OVS, *i.e.*, *user space* module and *kernel data path* module. The *kernel data path* module receives a packet from the network interface. If there are no rules matching this packet, it

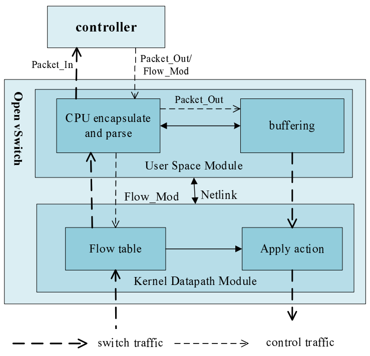


Fig. 1. Internal architecture of OpenvSwitch

will be sent to the CPU of *user space* module. The packet header is encapsulated into a *packet_in* message which is sent to the controller. The controller makes forwarding decision for the mismatched packet according to global network state information. Then the *packet_out* message and the *flow_mod* message are sent to OVS. The *user space* module parses the messages from the controller, forwarding the mismatched packet and updating flow table at the same time.

OVS is responsible for forwarding packets at data plane. OpenFlow protocol allows the controller to communicate with OVS, obtaining statistical information of flow table entries, and dynamically adding, updating, deleting flow forwarding rules in the switches. Thus the controller can monitor the state of the whole network. The *user space* module receives flow forwarding rules from the controller, matching flow table for all the received packets and forwarding them according to matching actions. OVS caches the results in the *kernel data path* module so as to realize fast forwarding of the subsequently arrival packets. It allows OVS to work independently of any SDN controller as it only needs to understand the OpenFlow protocol [16]. Through the above analysis, it can be seen that packet processing delay is mainly composed of the processing delay within OVS (including the flow table lookup delay and the delay of processing *packet_in* message, *packet_out* message and *flow_mod* message), the two-way propagation delay between the controller and the switch, and the delay of making decisions in the controller.

4 Queueing Network Model for OpenvSwitch

This section presents a mathematical theory analysis for packet processing in OVS. We analyze the working principle of OVS and establish the mathematical model, called *Model SC*, based on the queueing theory. In *Model SC*, "*S*" refers to packet processing in the switch and "C" refers to packet processing in the controller. Then, we improve *Model SC* and build a optimized queuing network model, which is called *Model MSC*. "*M*" means using optimized methods to build models, including multi-thread processing, priority queue settings and different packet scheduling policies. In addition, existing studies have shown that the built-in buffer of OVS can reduce the two-way propagation delay between the controller and the switch [20, 21], thus we model the buffer with queueing theory to investigate how buffer size affects the performance.

4.1 Queueing network model (*Model SC*) analysis

We build a queueing network model for OVS and analyze different influencing factors that affect packet processing delay. According to the working principle of OVS, packet processing in OVS is divided into four phases. As depicted in Fig.2, the switch is built into a queueing network model, i.e., *Model SC*, which can be decomposed into four subsystems, each of which is established as a queueing model. Detailed analysis of each phase is described as follows.

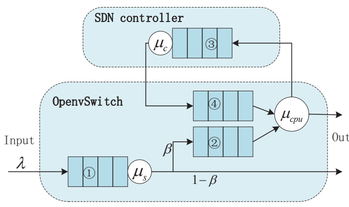


Fig. 2. The queueing network model for OpenvSwitch

Phase one: the lookup process within the flow table We simplify the lookup process of flow table and establish an M/M/1/K model, where K is the maximum capacity of the input queue of the port. We assume that packet arrivals obey *Poisson* distribution denoted by λ and time of flow table lookup obeys negative exponential distribution with the service rate denoted by μ_s. Due to the limitation of queue capacity, the arrival packets may get lost. The queue is a finite integer [17, 18] , so the loss probability could be calculated. The loss probability P_s at this phase is illustrated as equation (1).

$$P_s = (\frac{\lambda}{\mu_s})^K \tag{1}$$

Where, λ means the average packet arrival rate and μ_s is expressed as the average lookup rate of flow table in the switch.

This phase mainly conducts flow table lookup for the arrival packets. If the packets match forwarding rules, forwarding operations are executed directly. Otherwise, the packets are forwarded to the CPU of *user space* module for further processing. The average processing time of this phase is calculated as equation

(2) based on queueing theory.

$$d_{i,1} = \frac{1}{\mu_s - (1 - P_s)\lambda} \tag{2}$$

Phase two: the encapsulation of *packet_in* message If an arrival packet has not matched any rules of flow table, this packet is forwarded to the CPU of *user space* module. The header of this mismatched packet is extracted and encapsulated into a *packet_in* message which is sent to the controller for forwarding decisions. The probability of such a situation is denoted as table-miss rate represented by β. Therefore, we consider the process within the CPU as an M/M/1 queueing model, and assume that the average processing time at this phase obeys negative exponential distribution represented by $\beta\ \mu_2$. In addition, the CPU receives the control operation packets from the controller at the same time. It is assumed that the CPU adopts *First In First Out (FIFO)* strategy for packet processing.

Phase three: packet processing in the controller The controller has a global view and formulates forwarding decisions based on network state information. We simplify the process of the controller and mainly focus on the processing of *packet_in* messages. We establish a single queueing model for the controller. We assume that the controller has a queue with infinite capacity for *packet_in* messages. When a *packet_in* message arrives at the controller, the controller processes it with a *FIFO* queue. The queue in the controller is denoted as an M/M/1 queueing model. Existing studies have already investigated such kind of models for SDN controllers [12, 19]. The average processing delay of a *packet_in* message in the controller can be calculated by equation (3).

$$d_{i,3} = \frac{1}{\mu_c - \lambda_c} \tag{3}$$

Where λ_c refers to the average packet arrival rate, and μ_c refers to the average packet processing rate within the controller.

Phase four: parsing the messages from the controller After the controller parsing the *packet_in* message, it formulates forwarding decisions and sends a *packet_out* message and a *flow_mod* message) to the switch. The *packet_out* message instructs switch to directly forward the mismatched packet through a specified interface and the *flow_mod* message instructs switch to install, update or delete the forwarding rule in the flow table. At this phase, the CPU is mainly in charge of parsing the control operation messages from the controller. The packets are forwarded according to the parsing results. It is suited as an M/M/1 model, and the parsing process is assumed to follow the negative exponential distribution. The processing rate is denoted by μ_4.

Since phase two and phase four have roughly the same processing in the switch CPU, we can combine them into a queuing model. Note that the *FIFO)* queueing strategy is adopted. Hence the parameters of the queuing model meet the conditions as shown in equation (4).

$$\begin{cases} \lambda_{cpu} = \lambda_2 + \lambda_4 \\ \mu_2 + \mu_4 \le \mu_{cpu} \end{cases} \tag{4}$$

Where μ_{cpu} is the maximum processing capability of the switch CPU. Therefore, the average processing time within the switch CPU is calculated as equation (5).

$$d_{i,2} + d_{i,4} = \frac{1}{\mu_2 + \mu_4 - \lambda_{cpu}} \tag{5}$$

We consider propagation delay between the switch and the controller, because it is important to estimate packet processing delay. It consists of constant propagation delay and dynamic queuing delay. In this paper, we analyze the maximum propagation delay between the switch and the controller, *i.e.*, $d_t = \max\limits_{i=1,2,\ldots,n} \{d_{i,t}\}$.

In a real network, there is almost no queuing in the propagating process [13], so the queuing delay can be ignored. The maximum propagation delay between the switch and the controller can be considered a constant as shown in equation (6).

$$d_t = \max_{i=1,2,\ldots,n} \{d_{i,t}\} = constant \tag{6}$$

According to the queuing network models established above, we can estimate the processing delay of a packet, from arriving at the switch to being forwarded to the next hop successfully. The average packet processing delay D_{pkt} mainly consists of four parts which are described as equation (7).

$$D_{pkt} = \frac{1}{n} \sum_{i=1}^{n} [(1-\beta)d_{i,1} + \beta(\sum_{k=1}^{4} d_{i,k} + 2d_t)] \tag{7}$$

Where $d_{i,k}$ is the processing delay of the i_{th} packet at phase k.

4.2 Optimized queueing network model (*Model MSC*) analysis

To reduce packet processing delay, an optimized queueing model for OVS is proposed, i.e., *Model MSC*. Compared with *Model SC*, *Model MSC* mainly optimize flow table lookup and switch CPU processing with the methods of multi-threaded processing, priority queue settings and different queue scheduling policies. The optimized queueing network model is shown in Fig.3.

Optimizing flow table lookup To reduce the delay of flow table lookup, we use multi-threaded processing to achieve rapid flow table lookup at phase one. The switch CPU uses *FIFO* strategy to process packets. We define the queueing network model as *Model MSC-F* and "*F*" means the *FIFO* queue is utilized.

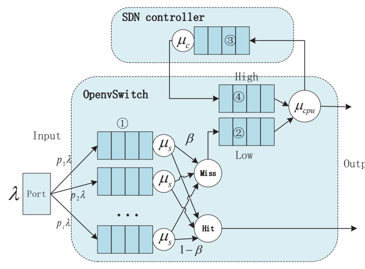


Fig. 3. The optimized queueing network model for OpenvSwitch

During the multi-thread process, each thread is established as an M/M/1/K model, so the multi-thread process can be regarded as an M/M/C/K queuing network model. When packets arrive at the switch, they will be assigned a thread with the probability of p_j meeting $\sum_{j=1}^{c} p_j = 1$. The probability each thread receiving packets follows the proportional relation of $\{p_1, p_2, ..., p_c\}$. The average processing delay of multi-thread process is depicted in equation (8).

$$d^{'}_{i.1} = \max_{j=1,2,...,m}\{\frac{1}{\mu_s - (1-P_s)p_j\lambda}\} \tag{8}$$

To further optimize the performance, we can use different scheduling policies to assign the arrival packets during the multi-thread processing. We conduct comparative analysis about this point in Section V.

Optimizing switch CPU processing We set the priority queues in the switch CPU, handling messages from the controller as quickly as possible to reduce the waiting delay of the packets buffered in the built-in buffer. Fig.3 shows the model which uses priority queues for switch CPU with finite capacity. We define this model as *Model MSC-P* and"*P*" refers to the priority queues designed in the switch CPU.

Priority queues isolate control packets from data packs. In *Model MSC-P*, packets received at phase two enter into the low-priority queue and packets received at phase four (*i.e.*, *packet_out* messages and *flow_mod* messages)) enter into the high-priority queue. The switch CPU can be regarded as the server of a queuing system. It prioritizes the packets in the high-priority queue. When there are no packets in the high-priority queue, it processes the packets in the low-priority queue. Therefore, the control packet arrival rate can be considered as the average packet arrival rate of switch CPU (i.e.,$\lambda^{'}_{cpu}$,). The average processing delay of switch CPU is represented by equation (9).

$$d^{'}_{i,2} + d^{'}_{i,4} = \frac{1}{\mu_{cpu} - \lambda^{'}_{cpu}} \tag{9}$$

Based on the above analysis, we can calculate the average processing delay with the optimized queueing network model.

$$D'_{pkt} = \frac{1}{n}\sum_{i=1}^{n}[(1-\beta)d'_{i,1}+\beta(\sum_{k=1}^{4}d'_{i,k}+2d_t)] \tag{10}$$

Where $d'_{i,k}$ is the processing delay of the i_{th} packet at phase k with *Model MSC*.

4.3 Built-in buffer queueing model analysis

The main purpose of setting up a built-in buffer is to store mismatched packets. It caches subsequent packets belonging to the same flow in the built-in buffer, which can reduce the number of *packet_in* messages and decrease the communication load between the controller and the switch. Generally, there are three main situations that need to be handled by the buffer.

(1) When receiving a mismatched packet, it is necessary to find out whether there are packets belonging to the same flow in the built-in buffer. If there are, the mismatched packet will be stored in the buffer. Otherwise, its header information is encapsulated into a *packet_in* message sent to the controller.

(2) When waiting for the controller to return the *packet_out* message, if a packet stored in the built-in buffer exceeds the time limit, proper processing is required. For example, the switch sends a *packet_in* message to the controller again or directly discards this packet.

(3) The controller makes a forwarding decision for the packets of a flow and generates a *packet_out* message sent to the switch. The switch parses the *packet_out* message and forwards the buffered packets of this flow to a specific interface according to the forwarding operation.

Existing studies apply the built-in buffer to reduce the interactions between the controller and the switch [20, 21]. When using the flow-level buffering mechanism [20], only one *packet_in* message is sent to the controller for all the mismatched packets of a flow. Experiment results show that the built-in buffer can reduce the packet processing delay effectively. In this paper, we model the built-in buffer based on queueing theory to investigate how buffer size affects the packet processing delay through a theoretical analysis.

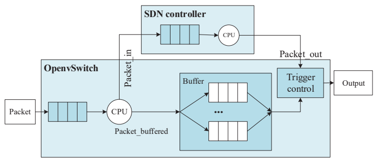


Fig. 4. The built-in buffer queueing model

We establish a triggered queueing network according to the processing mechanism of the built-in buffer. As shown in Fig.4, the process within the controller is simplified to an M/M/1 queueing model. The *packet_out* message from the controller is considered as the trigger information to release corresponding packets buffered in the built-in buffer.

In this model, we assume that the arrival process of mismatched packets follows *Poisson* distribution with the parameter of λ_{mis}. The average rate of packet arriving at the controller satisfies equation (11).

$$\lambda_{buf} = \omega \lambda_{mis} \tag{11}$$

Where ω denotes the proportion of *packet_in* messages generated for arrival packets.

The processing process of controller obeys the negative exponential distribution with the parameter of μ_{buf}. The average processing delay of controller can be calculated by equation (12).

$$d_{buf} = \frac{1}{\mu_{buf} - \lambda_{buf}} \tag{12}$$

The propagation delay of the communication channel between the switch and the controller is considered invariable. As shown in equation (13), the maximum propagation delay is adopted.

$$d_{s \to c} = d_{c \to s} = d_t \tag{13}$$

Where $d_{s \to c}$ denotes the delay between the switch sending the *packet_in* message and the controller receiving the *packet_in* message, and $d_{c \to s}$ denotes the delay between the controller sending the *packet_out* message and the switch receiving the *packet_out* message.

If there are no rules matching the packets of a flow, a series of operations are carried out for setting up this flow. the delay of setting up a flow starting from the first packet of the flow entering the switch to the packet leaving the switch. According to the above analysis, the delay of setting up a flow can be calculated as equation (14).

$$D_{buf} = d_{s \to c} + d_{buf} + d_{c \to s} = 2d_t + \frac{1}{\mu_{buf} - \lambda_{buf}} \tag{14}$$

The built-in buffer size depends on the number of arrival packets during the period of setting up flows. Hence the size of built-in buffer is expressed as equation (15).

$$B = \lambda_{pkt\_buf} D_{buf} \tag{15}$$

Where λ_{pkt_buf} is the average packet arrival rate at the built-in buffer. The packet processing delay (i.e., D_{pkt_buf}) with a built-in buffer can be calculated by equation (16).

$$D_{pkt\_buf} = \frac{1}{n} \sum_{i=1}^{n} [(1-\beta) d'_{i,1} + \beta (d'_{i,CPU} + D_{buf} + \frac{B}{\lambda_{release}})] \tag{16}$$

Where $d'_{i,CPU}$ denotes the packet processing delay of switch CPU and $\lambda_{release}$ denotes the average rate of releasing the packets buffered in the built-in buffer.

5 Performance evaluation

In this paper, a switch interacting with a controller is taken as the experimental scenario. The proposed queuing network models are carried out based on discrete event system simulation. We conduct a theoretical analysis of the performance influencing factors, including average packet arrival rate, flow table miss rate, packet scheduling policy and built-in buffer size.

The parameter settings for the proposed queuing network models are shown in Table I [9, 11]. The average packet arrival rate λ ranges from 1000 to 4000 pkts/sec and the flow table miss rate β ranges from 0.1 to 1. The flow table lookup rate μ_s is 2500 pkts/sec and the switch CPU processing rate μ_{cpu} is 2000 pkts/sec. The controller processing rate μ_c is 2500 pkts/sec and the packet loss rate P_s ranges from 0.01 to 0.1. Each experiment is executed for 20 times under different parameter settings. The average packet processing delay and the packet loss rate are calculated. How the influencing factors affect the performance is investigated with these analytical models, including *Model SC* which is the standard queueing network model for OVS, *Model MSC-F* which optimizes flow table lookup, and *Model MSC-P* which optimizes switch CPU processing.

Table 1. Parameter settings for the proposed queuing network models

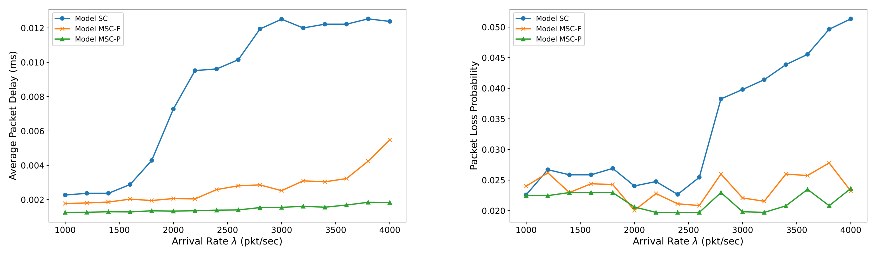
parameters	value
Table miss probability, β	0.1-1
Packet arrival rate, λ	1000-4000 pkts/sec
Lookup processing rate, μ_s	2500 pkts/sec
Switch CPU processing rate, μ_{cpu}	2000 pkts/sec
Controller processing rate, μ_c	2500 pkts/sec
Packet loss rate, P_s	0.01-0.1

5.1 Impact of packet arrival rate

Packet arrival rates may have a significant impact on the processing delay. Fig.5a shows the impact of arrival rate on average packet processing delay D_{pkt} of *Model SC*, *Model MSC-F* and *Model MSC-P*. For *Model SC*, packet arrival rate affects the the average processing delay greatly, and the value of D_{pkt} increases rapidly with the growth of λ. This is because the number of packets arriving at the switch increases with the increase of λ, resulting in a longer waiting time in the input queue.

Compared to *Model SC*, the average processing delay of *Model MSC-P* and *Model MSC-F* is less affected by the packet arrival rate. This is because multi-thread processing can accelerate packet processing and reduce the input queuing delay. At the same time, *Model MSC-P* prioritizes the decision messages from the controller, which reduces the waiting delay of mismatched packets. Therefore, the average packet processing delay of *Model MSC-P* is smaller than that of

Model MSC-P. We also find that *Model MSC-P* is basically not affected by the packet arrival rate.



(a) ATT (s) (b) AEC (J)

Fig. 5. Fig.5a: Average packet processing delay under different arrival rates, Fig.5b: Packet loss probability under different arrival rates.

Fig.5b shows the impact of arrival rate on packet loss probability, i.e., P_{s}. It can be observed that *Model MSC-P* and *Model MSC-F* have better performance on packet loss probability. P_{s} of *Model MSC-P* is smaller than that of *Model MSC-F* in most cases. Compared to *Model SC*, the packet loss probability of *Model MSC-P* and *Model MSC-F* is less affected by λ. This is attributed to the multi-thread processing mechanism and the priority queue in the optimized models. *Model SC* adopts the single-thread processing mechanism to process packets. When λ is not greater than μ_s, packet loss probability is low and fluctuates around 2.5%. This is because packets are processed quickly by the switch and there is no congestion. When λ is greater than μ_s, congestion occurs in the switch, making the packet loss probability increase. After that threshold, P_{s} of *Model SC* increases with the growth of λ.

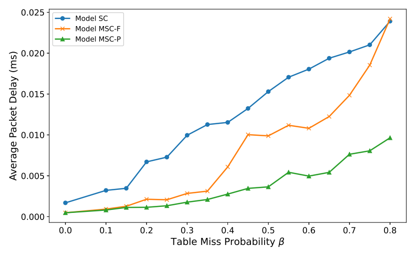
5.2 Impact of table miss probability

Table miss rate β is the ratio of mismatched packets to the total arrival packets. It is expressed by equation (17).

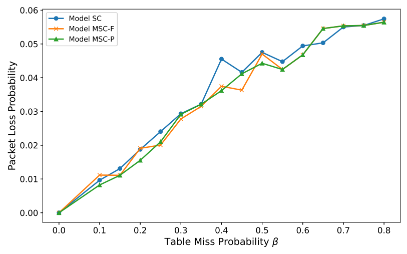
$$\beta = \frac{N_{pkt\_in}}{N_{total}} \tag{17}$$

Where N_{pkt_in} means the number of mismatch packets and N_{total} means the number of total arrival packets.

Fig.6a shows the impact of table miss probability on average packet processing delay. It can be seen that packet processing delay of the three queuing network models increases with the growth of table miss probability. Due to multi-thread processing, *Model MSC-F* and *Model MSC-P* perform better than



(a) ATT (s)



(b) AEC (J)

Fig. 6. Fig.6a: Average packet processing delay under different table miss probability, Fig.6b: Packet loss probability under different table miss probability β.

Model SC. In addition, *Model MSC-P* presents an obvious advantage compared with *Model SC* and *Model MSC-F*. This is also because *Model MSC-P* prioritizes the control messages from the controller and forwards the packets cached in the built-in buffer of the switch as soon as possible, which greatly reduces the waiting delay of mismatched packets. Therefore, *Model MSC-P* can effectively mitigate the impact of table miss probability on packet processing delay.

Fig.6b shows the changes of packet loss under different table miss probability. It can be seen that the packet loss rates of the three models increase with the growth of table miss probability. The values of them are similar to each other under the same flow table miss probability. That is to say *Model MSC-F* and *Model MSC-P* can not effectively reduce the packet loss. To sum up, when table miss rate is high, *Model MSC-P* has smaller packet processing delay and lower packet loss rate. In other words, it performs better in handling the mismatched packets.

5.3 Impact of packet scheduling policies

For the optimized models, the switch adopts the multi-thread processing mechanism to process the received packets. Each thread contains a *FIFO* queue with limited capacity. How to schedule the arrival packets for the parallel queues is worth discussing. In this section, we evaluate the impact of packet scheduling policies on processing delay. The following four packet scheduling policies are designed for the multi-thread processing mechanism of *Model MSC-F*.

Policy one When a packet arrives at the switch, the processing unit schedules the first input queue to process it. If this input queue is blocked by this packet, the processing unit will allocate subsequent packets to another queue. If the input queues of all threads are blocked, the processing unit will no longer receive packets for the switch.

Policy two The processing unit allocates arrival packets to input queues in turn for processing. The processing unit will schedule from queue one to queue M assuming that the interface has M queues in total. After polling M queues, The processing unit schedules from queue one.

Policy three Packets are allocated to input queues of multiple threads according to a specified probability. An arrival packet will be assigned to a queue of one thread with the probability of p_j, where $\sum_{j=1}^{c} p_j = 1$. The probability assigned to each thread follows a proportional relationship, i.e., $\{p_1, p_2, ..., p_c\}$. The processing unit distributes the arrival packets to the threads for processing according to the proportional relationship.

Policy four When a packet arrives at the switch, processing unit randomly assigns it to an input queue for processing. The input queue of each thread is equally to be selected.

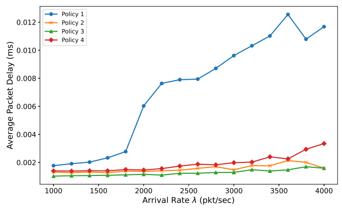


Fig. 7. Average packet processing delay of different scheduling policies

Fig.7 shows the impact of packet scheduling polices on processing delay across different arrival rates. Results show that packet processing delay of policy one increases with the growth of arrival rate. When $\lambda \leq \mu_s$, the average packet processing delay is low and keeps stable. When $\lambda > \mu_s$, the packet arrival rate exceeds the switch processing rate. As a result, the average packet processing delay presents a rapid growth with the increase of arrival rate. Packet processing delay of the other three policies is not significantly affected by the arrival rate. Switch randomly assigns packets to threads, which can alleviate the congestion effectively. We find that policy three has the lowest processing delay in most cases. This is because the switch distributes packets to threads according to the pre-defined probability relationship. This makes full use of multi-thread resources, which can reduce the processing delay of the switch.

5.4 Impact of built-in buffer size

Built-in buffer is mainly used to store mismatched packets. When the switch receives the first data packet of a new flow, the switch CPU encapsulates the

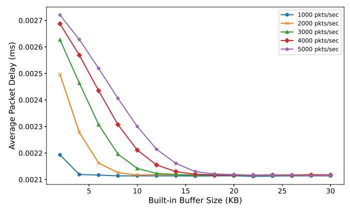


Fig. 8. flow setup delay

header information of the packet and sends a *packet_in* message to the controller, while the subsequently arrival packets of this flow will be buffered in the built-in buffer, waiting for the decision issued by the controller. When the built-in buffer is exhausted, the switch CPU will encapsulate the whole mismatched packet into *packet_in* message and send it to the controller. After making a decision, the controller will generate *packet_out* message and *flow_mod* message, which are sent to the switch. This will lead to an increase in the load of the communication channel between the switch and the controller, resulting in a rapid decline in the processing performance of the switch. Therefore, avoiding the exhaustion of the built-in buffer is the key to reduce the switch processing delay. Hence it is necessary to determine the size of the built-in buffer to reduce its impact on the processing delay.

By setting different built-in buffer sizes and carrying out several experiments under different packet arrival rates, we investigate the changes of processing delay and then determine the range of the built-in buffer size. Since the size of packets arriving at the switch may be different, a fixed storage space is allocated for each buffered packet in order to store mismatched packets conveniently. Fig.8 shows the packet processing delay of different flows across different built-in buffer sizes.

It can be seen that the average processing delay decreases obviously with the increase of the buffer size. However, when buffer size exceeds a certain value, its impact on packet processing delay becomes smaller until it remains basically unchanged. In addition, the impact of arrival rate on processing delay decreases with the increase of the buffer size. When the size of built-in buffer is constant, the processing delay increases with the growth of arrival rate. However, when buffer size exceeds 20 KB, the processing delay will no longer be affected by the arrival rate. Therefore, the built-in buffer of the switch can alleviate the impact of arrival rate on processing delay to a certain extent. Therefore, we can set the built-in buffer size for the switch according to the experiment results, so as to reduce the processing delay of the switch and improve the performance of the whole network.

6 Conclusion

In this paper, we investigate and optimize the performance of OpenvSwitch with queueing theory. First of all, we present a basic queueing network model according the working principles of OVS. We can evaluate the performance influencing factors with the proposed model. In addition, we optimize the basic model from many aspects, including multi-threaded processing, setting priority queues and utilizing different packet scheduling policies. Then we expound the impact of built-in buffer size on packet processing delay.

We evaluate these optimized methods through a comparison analysis. We compare the performance of *Model SC* (basic model), *Model MSC-F* (multi-thread with *FIFO* queue model) and *Model MSC-P* (multi-thread with priority queue model). Results reveal that the optimized models have better performance than the basic model. We also find that the built-in buffer size has a more significant impact on packet processing delay via theoretical verification. In the future, we will extend the use of the mathematical analysis model to evaluate and optimize the performance of SDN switches, and provide guidelines for optimal switch design.

7 Acknowledgements

The authors would like to thank Dr. Qing Li for his kind help and constructive comments. This work is supported by the National Key Research and Development Program of China under Grant No. 2019YFB1802600; the National Natural Science Foundation of China under Grant No. 62072091; the Fundamental Research Funds for the Central Universities under Grant No. N2016005; Shenzhen Development and Reform Commission No.XMHT20200105010; Beijing Natural Science Foundation under Grant No. M21030; Open Research Fund Program of Beijing National Research Center for Information Science and Technology.

References

1. W. Xia, Y. Wen, C. H. Foh, D. Niyato and H. Xie, "A Survey on Software-Defined Networking," in IEEE Communications Surveys Tutorials, 2015, vol. 17, no. 1, pp. 27-51.
2. K. Benzekki, A. E. Fergougui, A. E. Elalaoui, "Software defined networking (SDN): a survey," in Security and Communication Networks, 2016, vol.9, no.18, pp. 5803-5833.
3. OVS, ¡°Open vswitch manual,¡± accessed 16 May 2017. [Online]. Available: http://www.openvswitch.org/
4. B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, ¡°The design and implementation of Open vSwitch,¡± in 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), Oakland, CA, 2015, pp. 117¨C130.

5. S. Azodolmolky, R. Nejabati, M. Pazouki, P. Wieder, R. Yahyapour and D. Simeonidou, "An analytical model for software defined networking: A network calculus-based approach," in 2013 IEEE Global Communications Conference (GLOBECOM), Atlanta, GA, 2013, pp. 1397-1402.
6. M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll and P. Tran-Gia, "Modeling and performance evaluation of an OpenFlow architecture," in 2011 23rd International Teletraffic Congress (ITC), San Francisco, CA, 2011, pp. 1-7.
7. OpenFlow Switch Specification, accessed 20 June 2018. [Online]. Available: https://www.opennetworking.org/
8. J. Ansell, WKG. Seah , B. Ng, et al. "Making queueing theory more palatable to SDN/OpenFlow-based network practitioners," in Network Operations and Management Symposium. IEEE, 2016, pp. 1119-1124.
9. S. Muhizi, G. Shamshin, A. Muthanna, et al. "Analysis and Performance Evaluation of SDN Queue Model," in Wired/Wireless Internet Communications, 2017, pp. 26-37.
10. Z. Shang, K. Wolter, "Delay evaluation of openflow network based on queueing model," in arXiv preprint arXiv:1608.06491, 2016.
11. Wang Zhengqiang, Zhao Shuyi, Fan Zifu, Wan Xiaoyu, "Performance Modeling and Analysis of Control Plane for SDN Based on Queuing Theory," in Wireless Personal Communications. 2017, PP. 1-11.
12. K. Mahmood, A. Chilwan, O. sterb and M. Jarschel, "Modelling of OpenFlow-based software-defined networks: the multiple node case," in IET Networks, 2015, vol. 4, no. 5, pp. 278-284.
13. M. Haiyan, Y. Jinyao, P. Georgopoulos and B. Plattner, "Towards SDN based queuing delay estimation," in China Communications, vol. 13, no. 3, pp. 27-36, March 2016.
14. D. Singh, B. Ng B, Y. C. Lai, et al. ¡°Modelling Software-Defined Networking: Switch Design with Finite Buffer and Priority Queueing," in IEEE, Conference on Local Computer Networks. IEEE Computer Society, 2017, pp. 567-570.
15. A. Fahmin, Y. C. Lai, M. S. Hossain, Y. D. Lin and D. Saha, "Performance Modeling of SDN with NFV under or aside the Controller," 2017 5th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW), Prague, 2017, pp. 211-216.
16. B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, ¡°The design and implementation of Open vSwitch,¡± in 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), Oakland, CA, 2015, pp. 117¨C130.
17. R. J. Simcoe, J. P. Robert. "Perspectives on ATM switch architecture and the influence of traffic pattern assumptions on switch design," in ACM SIGCOMM Computer Communication Review. 1995, vol. 25, no.2, pp. 93-105.
18. S. C. Liew, "Performance of various input-buffered and output-buffered ATM switch design principles under bursty traffic: simulation study," in IEEE Transactions on Communications, vol. 42, no. 234, pp. 1371-1379, Feb/Mar/Apr 1994.
19. G. Wang, J. Li and X. Chang, "Modeling and performance analysis of the multiple controllers' approach in software defined networking," 2015 IEEE 23rd International Symposium on Quality of Service (IWQoS), Portland, OR, 2015, pp. 73-74.
20. F. Li, J. Cao, X. Wang, et al. "Applying Buffer to SDN Switches: Benefits Analysis and Mechanism Design," in IEEE Transactions on Cloud Computing, 2018.
21. Li, F., Cao, J., Wang, X., Sun, Y., Pan, T., Liu, X. (2017). Adopting SDN Switch Buffer: Benefits Analysis and Mechanism Design. IEEE, International Conference on Distributed Computing Systems (pp.2171-2176). IEEE.