Computer Networks 102 (2016) 172–185

Contents lists available at ScienceDirect

Computer Networks

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

Performance evaluation of OpenFlow-based software-defined networks based on queueing model

Bing Xiong[a,*], Kun Yang[b], Jinyuan Zhao[c], Wei Li[a], Keqin Li[d]

[a] School of Computer and Communication Engineering, Changsha University of Science and Technology, Changsha 410114, PR China
[b] School of Software, Northeast Normal University, Changchun 130024, PR China
[c] School of Software, Central South University, Changsha 410075, PR China
[d] Department of Computer Science, State University of New York at New Paltz, New York 12561, USA

ARTICLE INFO

Article history:
Received 27 August 2015
Revised 17 January 2016
Accepted 7 March 2016
Available online 4 April 2016

MSC:
00-01
99-00

Keywords:
Software-defined networking
Performance evaluation
Queueing model
SDN controllers
OpenFlow switches

ABSTRACT

OpenFlow is one of the most famous protocols for controller-to-switch communications in software-defined networking (SDN), commonly seen as a promising way towards future Internet. Understanding the performance and limitation of OpenFlow-based SDN is a prerequisite of its deployments. To achieve this aim, this paper proposes a novel analytical performance model of OpenFlow networks based on queueing theory. After depicting a typical network scenario of OpenFlow deployments, we model the packet forwarding of its OpenFlow switches and the packet-in message processing of its SDN controller respectively as the queueing systems $M^X/M/1$ and $M/G/1$. Subsequently, we build a queueing model of OpenFlow networks in terms of packet forwarding performance, and solve its closed-form expression of average packet sojourn time and the corresponding probability density function. Finally, the numerical analysis is carried out to evaluate our proposed performance model with different parameter values. Furthermore, our controller model is contrasted with the classical one by utilizing the popular benchmark Cbench. Experimental results indicate that our controller model provides a more accurate approximation of SDN controller performance.

1. Introduction

As a new network paradigm, Software-defined Networking (SDN) is currently seen as one of the promising approaches in the way towards future Internet [1–3]. It decouples the network control out of forwarding devices, and allows for a separate controller entity that may change the forwarding rules in modern switches [4]. The separation of the control logic from the forwarding substrate greatly facilitates the deployment and operation of new services and enables SDN to react gracefully to the change of network demands and modifications to the substrate topology [5]. These pave the way for a more flexible, programmable, and innovative networking [6,7].

In SDN architecture, packet forwarding devices are manipulated by the logically centralized controller through south-band interface, typically the OpenFlow protocol [8,9]. This architecture raises the performance bottlenecks of the controller and its capacity to handle all OpenFlow switches, especially for large and highly distributed networks [10]. Currently, OpenFlow-based SDN concept is finding its way into commercial applications [11,12], and a growing number of experiments over SDN-enabled networks are expected. This will create new challenges, as the questions of SDN performance and scalability have not yet been fully investigated in recent researches.

[*] Corresponding author. Tel.: +86 18773116229; fax: +86 0731 85258462.
E-mail address: xiongbing@csust.edu.cn, xiongbing.csust@qq.com, xiongbing.hust@qq.com (B. Xiong).

http://dx.doi.org/10.1016/j.comnet.2016.03.005
1389-1286/© 2016 Elsevier B.V. All rights reserved.

Understanding the performance and limitation of OpenFlow-based SDN concept is a prerequisite for its usage in practical applications. Specifically, an initial estimate of the performance of OpenFlow networks is essential for network architects and designers. Although simulation studies and experimentation are among the widely used performance evaluation techniques, analytical modeling has its own benefits. A closed-form description of a networking architecture enables the network designers to have a quick approximate estimate of the performance of their design, without the need to spend considerable time for simulation studies or expensive experimental setups.

To the best of our knowledge, there are very few research activities to analytically model and evaluate OpenFlow-based SDN performance. Some researchers [13–15] utilized network calculus as a mathematical model to evaluate the performance bounds of controllers and switches in the worst case. Different from them, we focus on the average packet forwarding performance of OpenFlow networks at its equilibrium state. More close to our work, Zuo [19] and Yao [20] et al. model SDN controller performance with a $M^k/M/1$ queue by considering the flow setup requests to the controller as a batch arrival process. However, the batch arrival could not exactly characterize the pattern of flow setup requests from multiple switches. The closest work is the analytical model of OpenFlow-based SDN proposed by Jarschel et al., which approximates the data plane as an open Jackson network with the controller also modeled as a $M/M/1$ queue [17,18]. Nevertheless, the assumptions of their queueing models were a bit far away from network traffic measurements.

For the above situations, this paper is motivated to provide an accurate performance model of OpenFlow-based SDN with multiple OpenFlow switches. To achieve this aim, we first give a typical network scenario of OpenFlow deployments. Then, we investigate into the arrival process and processing time of its OpenFlow switches and SDN controller to reach their queueing models. Subsequently, we further build a queueing model of OpenFlow networks to characterize its packet forwarding performance, and solve its average packet sojourn time and probability density function. Finally, we perform numerical analysis on the queueing model in terms of different performance parameters, and our controller performance model is also evaluated under various network scenarios by using the prevalent benchmark Cbench.

With the above methodology, this paper aims to achieve the following conclusions as the main contributions: (a) pointing out that the queueing system $M^X/M/1$ is suitable to characterize the packet switching performance of an OpenFlow switch in our network scenario; (b) concluding that the packet-in message processing performance of SDN controller can be modeled as a $M/G/1$ queue; (c) proposing a queueing model for the packet forwarding performance of OpenFlow networks, and solving the closed-form expression of its average packet sojourn time and probability density function.

The rest of the paper is organized as follows. Section 2 introduces related work. In Section 3, we give a typical network scenario of OpenFlow deployments. Section 4 characterizes the packet switching performance of an OpenFlow switch as the queueing system $M^X/M/1$ after the investigations of its packet arrival process and packet switching procedure. In Section 5, we model the packet-in message processing performance of SDN controller as a $M/G/1$ queue based on the investigation of the message arrival process. Section 6 builds a queueing model of the packet forwarding performance through an OpenFlow switch, and solves its performance parameters including average packet sojourn time. In Section 7, we evaluate the queueing model with different performance parameters by utilizing numerical analysis, and our controller performance model under various network scenarios with the prevalent benchmark Cbench. Section 8 concludes the paper.

2. Related work

Software-Defined Networking (SDN) is an emerging paradigm that promises to change the limitations of current network infrastructures, by separating the network's control logic from the underlying routers and switches, promoting logical centralization of network control, and introducing the ability to program the network [21–23]. By this way, SDN offers flexible, dynamic, and programmable functionality of network systems, as well as many other advantages such as centralized control, reduced complexity, better user experience, and a dramatic decrease in network systems and equipment costs [24]. However, these advantages come with non-negligible penalty to essential network performance such as packet processing speed and throughput [25], which attributes to the involvement of a remote system called a controller in administration of all forwarding devices.

A controller in SDN paradigm is like an operating system for computers, which administrates its subordinate switching devices and provides a programmatic interface for user-written network applications. Thus it plays a critical role in the SDN architecture and has a significant impact on the entire network [26]. However, the current state of the SDN controller market may be compared to the extreme diversity of early operating systems for mainframes. Until now, there are more than 30 different controllers, created by different vendors/universities/research groups, written in different languages, and using different multi-threading techniques [32]. These differences make each controller with its own performance characteristics and specific controllers better suited for certain scenarios than others. Thus, it is necessary to understand the performance of the controllers and its impact factors, when planning a SDN deployment.

In [27], Jarschel et al. first developed a flexible OpenFlow controller benchmark Cbench to understand the performance characteristics of different controller implementations. The benchmark allows the emulation of scenarios and topologies by creating a set of message-generating virtual switches and configuring them independently from each other. By this way, it can be employed to carry out the measurement experiments of the controller performance on a per-switch basis, to reach a fine-grained analysis of controller performance bottlenecks. Tootoonchian et al. utilized the benchmark Cbench to measure several performance aspects of 4 publicly-available OpenFlow controllers (NOX [28], NOX-MT, Beacon [29], and Maestro [30]), including the minimum and maximum controller response time, maximum throughput, and the throughput and latency of the controller with a bounded number of packets on the fly [31]. Their measurement results achieve the conclusion that a single physical controller is not enough to manage a sizeable network.

In [32], Shalimov et al. further developed a more capable framework hcprobe. The framework can be used to create a set of advanced testing scenarios and provide the fine-tuning of measurement parameters to reach an insight into the controller scalability, reliability and security issues. With the framework, they provided a comprehensive performance analysis of popular open-source SDN/OpenFlow controllers (NOX [28], Beacon [29], Maestro [33]), Floodlight, POX, MuL, Ryu, which concluded that modern SDN/OpenFlow controllers were not ready for production networks. Shah et al. also evaluated the performance of four prominent OpenFlow controllers: NOX [28], Beacon [29], Maestro [33] and Floodlight under different metrics including thread scalability, switch scalability and message processing latency, which aimed to identify key performance bottlenecks and good architectural choices for designing OpenFlow based SDN controllers [34].

The concurrent operation of SDN switches with diverse capacities for control message processing leads to highly variable latencies for flow installations and modifications. To address this issue, Bozakov and Rizk used a queueing model to characterize the behavior of the control interface between the controller and a switch in terms of the amount of serviced messages over different time scales, and provided a measurement-based approach to derive an estimate of the corresponding service curves [13]. They also proposed a simple interface extension for controller frameworks which enables operators to configure time delay bounds for transmitted control messages. In [14,15], Azodolmolky et al. presented a mathematical framework based on network calculus to report the performance of the scalable SDN deployments. Given the parameters of the cumulative arriving process and the flow control functionality of the SDN controller, the network architect or designer is able to compute an upper bound estimate of the delay and buffer requirements of SDN controllers. Besides, Osgouei et al. proposed an analytical performance model of virtualized SDNs using network calculus to calculate the upper bounds of the latency of virtualized SDN controller and the service curve of each virtual network [16].

Closely related to our work, Jarschel et al. derived a basic analytical model based on $M/M/1$ queues to estimate the packet sojourn time and probability of lost packets for the network scenario, where a SDN controller is responsible for only a single OpenFlow switch in the data plane [17]. As a further step, they addressed the challenge of the case with multiple switches by approximating the data plane as an open Jackson network with the controller also modeled as a $M/M/1$ queue [18]. Nevertheless, their model was lack of the support of the measurements on the packet arrival process at the switch and packet-in message arrivals at the controller. Furthermore, Zuo et al. evaluated the queueing delay of flow setup requests in the control plane by introducing the multiple arrivals and single departure queue model [19], but did not reach a precise delay estimate. Yao et al. also modeled the flow setup requests to the controller as a batch arrival process to analyze the controller performance with a $M^k/M/1$ queue [20]. However, the batch arrival could not exactly characterize the pattern of flow setup requests from multiple switches. Thus, we are motivated to build a better performance model of OpenFlow network based on the investigations of the packet arrival process at the switch and packet-in message arrivals at the controller.

3. OpenFlow network deployments

As a novel network architecture, SDN enables researchers to test new ideas under realistic conditions on an existing network infrastructure. To be able to take action in the switching, OpenFlow separates the control plane from the data plane and connects them by an open interface, the OpenFlow protocol. As for OpenFlow deployments, a typical network scenario is depicted at Fig. 1.

As illustrated in Fig. 1, the OpenFlow network can be distinguished into the edge and the core ones. The edge network consists of many independent LANs connecting hosts, terminals and servers. Each LAN connects the core network via an access switch. The core network switches packet traffic among LANs and the Internet. To support OpenFlow, all switches in the core network are connected to a SDN controller directly or via other switches.

When a packet from a LAN arrives at its access switch in the core network, the switch performs lookups in its internal flow tables. If the lookup hits a table entry other than table-miss, the switch will forward the packet to the next one in a conventional way. Otherwise, the packet is supposed to belong to a new flow. In such case, the switch requests the controller for instructions by sending a packet-in message in encapsulation of the packet information. The controller determines the respective flow rule and installs it into all switches among the flow path. After that, all packets within the flow is correctly forwarded to their destination without requesting the controller.

In OpenFlow networks, a SDN controller is usually implemented as a network operation system, and responsible for multiple OpenFlow switches. All new flows from a LAN trigger its access switch to send a sequence of packet-in messages to the controller. These packet-in messages from all switches usually form up a waiting queue in the controller. Meanwhile, each switch keeps a packet queue at each ingress port. In consequence, we can analytically evaluate the performance of OpenFlow networks with queueing models.

4. Queueing model of OpenFlow switches

This section investigates into the packet arrival process and packet forwarding procedure at an OpenFlow switch, and characterize its packet forwarding performance with the queueing model $M^X/M/1$.

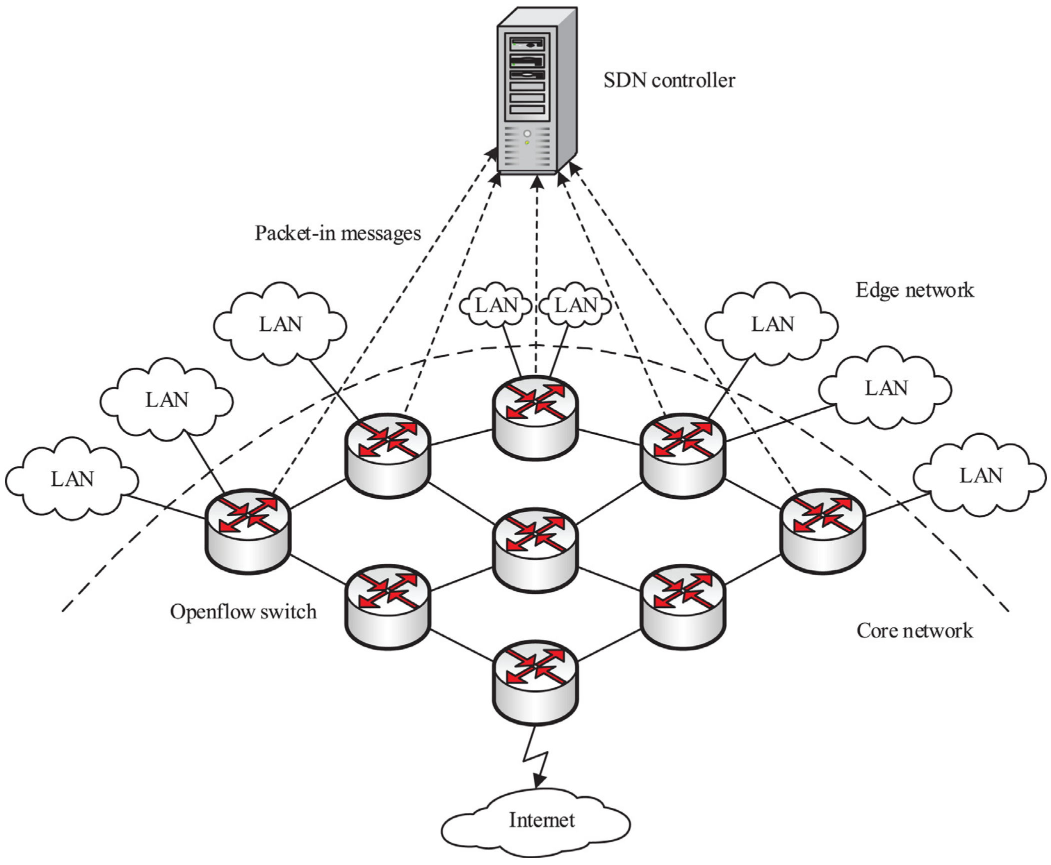


Fig. 1. A typical OpenFlow network scenario.

4.1. Queueing model analysis

Up to now, almost all analytical models on OpenFlow networks suppose that packet traffic into a switch conforms to the Poisson distribution. However, previous studies has pointed out that packet arrivals in computer networks is apt to be like trains [35,36] rather than Poisson stream [37]. The phenomenon of packet groups is a natural artifact of the protocols and applications for data transmission [38]. Firstly, the most widespread Internet application, i.e., web service, usually manifests as a mass of file downloading events in the viewpoint of networks. Secondly, the popular peer-to-peer file sharing produces bulk data transfer behaviors in network traffic. Lastly, the increasing streaming media applications, such as Internet telephony and television, generates persistent data downloading activities [39,40]. In summary, current Internet applications results in packet batch arrivals in packet switching networks.

Fig. 2 demonstrates the forwarding procedure of a packet in an OpenFlow switch. On receiving a packet, the switch puts it into the packet queue of its ingress port. As for its processing, the switch first retrieves it from the queue, and extracts its matching fields from all protocol headers to compute its flow key. After that, the key is used to look up the flow tables to match an entry. If the match fails, the switch fires off a packet-in message containing the full packet or its buffer ID to the connected SDN controller. The handling rule of the flow that the packet belongs to is learned in the controller, and is sent down to the switch to be added into its flow tables. When an entry other than table-miss is found, the packet is switched to the port in the entry via backplane, and wait for forwarding in the respective egress queue.

As for the packet forwarding procedure shown in Fig. 2, the flow table lookup dominates the processing time of a packet in an OpenFlow switch, as the backplane switching has completely broken the limitation of shared bandwidth by using CrossBar fabric. Since the table lookups for all packets are independent from each other, the packet processing time can be supposed as a random variable with negative exponential distribution. With the assumption of sufficient packet buffer for all ingress queues in the switch, the packet forwarding performance of an OpenFlow switch can be modeled as a $M^X/M/1$ queue. The queue can be characterized as the following assumptions: (a) packet batches arrive at the ith OpenFlow switch as Poisson stream with the rate $\lambda_i^{(b)}$; (b) the number of packets in a batch conforms to Poisson distribution with the parameter $\lambda_i^{(p)}$; (c) the packet processing time of the switch

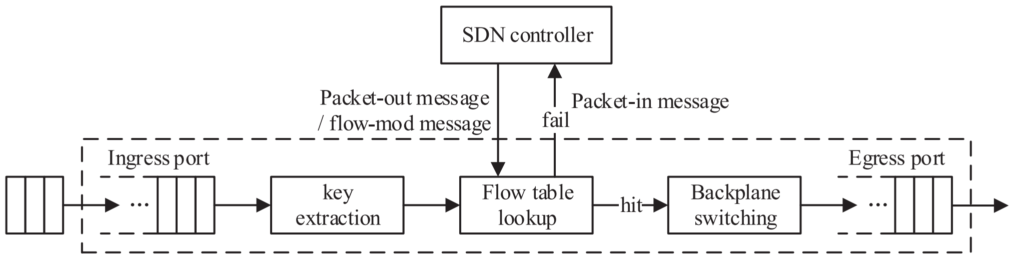


Fig. 2. The packet forwarding of an OpenFlow switch.

conforms to negative exponential distribution with the parameter $\mu_i^{(s)}$.

4.2. Queueing model solution

Suppose there is a batch with m packets arriving at the ith $(1 \le i \le k)$ OpenFlow switch, where n packets are waiting in its ingress queue for processing. Then, the lth arriving packet has to wait until the n waiting ones and the front $(l-1)$ ones have been processed. Its sojourn time in the switch can be expressed as the total processing time of $(n+l)$ packets in (1). Then, we can further compute the average sojourn time of the packet batch is in (2) and that of a packet in the switch in (3).

$$W_i^{(s)}(n,m,l) = \frac{n+l}{\mu_i^{(s)}}, 1 \le l \le m. \tag{1}$$

$$W_i^{(s)}(n,m) = \frac{1}{m}\sum_{l=1}^{m} W_i^{(s)}(n,m,l) = \frac{n}{\mu_i^{(s)}} + \frac{m+1}{2\mu_i^{(s)}}. \tag{2}$$

$$\begin{aligned}\overline{W_i^{(s)}} &= \sum_{n=0}^{\infty}\sum_{m=0}^{\infty} W_i^{(s)}(n,m)p_m p_n \\ &= \frac{1}{\mu_i^{(s)}}\sum_{n=0}^{\infty} np_n + \frac{1}{2\mu_i^{(s)}}\left(\sum_{m=0}^{\infty} mp_m + 1\right).\end{aligned} \tag{3}$$

According to the above assumptions, the number of packets in a batch m conforms to Poisson distribution with the parameter $\lambda_i^{(p)}$. Thus there exist (4). Besides, we can get the relationship between average queue length and average sojourn time in (5). Substituting (4) and (5) into (3), we can solve the average sojourn time of a packet in the switch in (6). Finally, we can also get the average queue length of the switch in (7) by putting (6) into (5).

$$\sum_{m=1}^{\infty} mp_m = \lambda_i^{(p)}. \tag{4}$$

$$\overline{L_i^{(s)}} = \sum_{n=0}^{\infty} np_n = \lambda_i^{(b)}\lambda_i^{(p)}\overline{W_i^{(s)}}. \tag{5}$$

$$\overline{W_i^{(s)}} = \frac{\lambda_i^{(p)}+1}{2(\mu_i^{(s)} - \lambda_i^{(b)}\lambda_i^{(p)})}. \tag{6}$$

$$\overline{L_i^{(s)}} = \frac{\lambda_i^{(b)}\lambda_i^{(p)}(\lambda_i^{(p)}+1)}{2(\mu_i^{(s)} - \lambda_i^{(b)}\lambda_i^{(p)})}. \tag{7}$$

5. Queueing model of SDN controllers

This section investigates into the arrival process and serving process of packet-in messages in a SDN controller, and characterize SDN controller performance with the queueing model $M/G/1$.

5.1. Packet-in message arrivals

An OpenFlow switch sends a packet-in message as flow setup request to its SDN controller on the arrival of a new flow. This implies that packet-in message stream from a switch to its controller has a one-to-one correspondence with flow arrival process. Network traffic measurements have indicated that flow arrival process in packet switching networks conforms to Poisson distribution [41,42]. In OpenFlow network deployments, a controller is usually responsible for multiple switches, and receives a stream of packet-in messages from each of them. Then, we can conclude that all packet-in messages at the controller from its switches constitute a Poisson stream in terms of the following theorem.

Theorem 5.1. *As for a SDN controller in charge of k OpenFlow switches in reactive mode, if flow arrival process at the ith* $(i = 0, 1, 2, \ldots, k)$ *switch is a Poisson stream with the parameter* $\lambda_i^{(f)}$ *independent of those at other switches, then all packet-in messages from the k switches to the controller conform to Poisson distribution with the parameter* $\lambda_{(c)}$ *in (8).*

$$\lambda_{(c)} = \sum_{i=1}^{k} \lambda_i^{(f)}. \tag{8}$$

Proof 5.1. Let $F_{i,j}(t)$ and $M_{i,j}(t)$ $(i = 0, 1, 2, \ldots, k, j = 0, 1, 2, \ldots, t \ge 0)$ respectively be the jth flow coming into the ith switch at the time t and the jth packet-in message arriving at the controller from the ith switch at the time t. On receiving a new flow $F_{i,j}(t)$, the switch looks up the flow tables in failure and sends a packet-in message to the controller. The controller receives the message at the time $(t + \Delta t)$.

The time difference Δt consists of failed lookup time of the flow tables, the transmission time of the packet-in message from the switch to the controller, and other negligible time such as forwarding time on switching backplane and scheduling time among output links in the switch. The failed lookup time chiefly depends on flow table size, which comes to be steady soon in network operations. The transmission time lies on the distance between the switch and the controller, which is fixed in a given network. In summary, Δt tends to be constant for all packet-in messages from a certain switch to its controller.

Consequently, it can be referred that the packet-in message stream $\{M_{i,j}(t), j = 1, 2, 3, \ldots\}$ from the ith switch to the controller is equivalent to the flow arrival process $\{F_{i,j}(t), j = 1, 2, 3, \ldots\}$ at the switch in the sense of stochastic process. Owing to the assumption that the process $\{F_{i,\,j}(t)\}$ at the ith switch conforms to Poisson distribution with the parameter $\lambda_i^{(f)}$ and those at all the k switches are independent of each other, we can further conclude that packet-in message stream $\{M_{i,\,j}(t)\}$ from the ith switch conforms to Poisson distribution with the same parameter in (9) and those from all the switches are independent of each other.

$$\{M_{i,j}(t)\} \leftrightarrow \{F_{i,j}(t)\} \sim P\left(\lambda_i^{(f)}\right). \tag{9}$$

With the above hypothesis, we can achieve the following conclusion that the synthetic packet-in messages at the controller from all the k switches $\sum_{i=1}^{k} M_{i,j}(t)$ confirms to Poisson distribution with the parameter $\lambda_{(c)} = \sum_{i=1}^{k} \lambda_i^{(f)}$ in (8) in terms of the composition theorem of Poisson stream. Thus, the above theorem is proved.

$$\left\{\sum_{i=1}^{k} M_{i,j}(t)\right\} \sim P\left(\sum_{i=1}^{k} \lambda_i^{(f)}\right). \tag{10}$$

□

5.2. Packet-in message processing

On the arrival of a packet-in message, the controller determines the rule of handing the respective flow and sends back the flow rule as a response to the packet-in message. Fig. 3 illustrates the packet-in message processing of a SDN controller. As shown in Fig. 3, the controller first caches all arrived packet-in messages from its substrate OpenFlow switches in a queue in order of their arrival time. The processing unit of the controller reads in a packet-in message from the queue after completing the handling of the last message. Then, the packet-in message is processed by looking up the forwarding tables commonly called the forwarding information base (FIB). The FIB is generated from the routing information base (RIB) at its consistent and stable state. The RIB contains information about the network topology built by path computation and topology discovery through the running of routing protocols or manual programming. Finally, the controller encapsulates the found forwarding rule in a packet-out or flow-mod message to the respective switch.

As seen from the above packet-in message processing of a SDN controller, the processing time of a packet-in message is chiefly derived from the FIB lookups. The FIB is generally stored with trie trees and its lookups are performed with longest prefix matching. So the comparison times of the FIB lookup are reckoned to distribute normally, and we can assume that the FIB lookup time conforms to normal distribution [43] with the position parameter $1/\mu_{(c)}$ and the scale parameter $\sigma_{(c)}$. The parameter $\mu_{(c)}$ represents the expectation of the processing rate of packet-in messages in the controller, and the parameter $\sigma_{(c)}$ denotes the standard deviation expectation of actual processing time to the expected one of packet-in messages in the controller.

5.3. Queueing model description

With the above queueing analysis, we can characterize the packet-in message processing of a SDN controller with the $M/G/1$ queueing model: (a) the arrival process of packet-in messages at the controller conforms to Poisson distribution with the parameter $\lambda_{(c)}$; (b) the processing time of packet-in messages in the controller conforms to normal distribution with the position parameter $1/\mu_{(c)}$ and the scale parameter $\sigma_{(c)}$; (c) there is a processing unit in the controller to handle packet-in messages with the first-come-first-serving principle, and the arrivals and processing of packet-in messages are independent of each other.

Suppose that $N(t)$ stands for the length of the packet-in message queue in the controller at time t. As for any time t, if there is a packet-in message being processed, the distribution of the rest processing time from the time t is no longer of memoryless property, and the queue length process $\{N(t), t \geq 0\}$ is no longer of Markov property. Suppose X_n as the number of packet-in messages in the controller at the time that the processing of the nth message is finished, it can be proved that $\{X_n, n \geq 1\}$ is a Markov chain, particularly called the imbedded Markov chain of the queue length process $\{N(t), t \geq 0\}$.

Suppose that T_n represents the required processing time of the $(n + 1)$th message from the time that the nth message leaves, we can express its probability density function in (11) in terms of its normal distribution assumption in the above queueing model.

$$g(x) = \frac{1}{\sqrt{2\pi}\,\sigma_{(c)}} e^{-\frac{(x-1/\mu_{(c)})^2}{2\sigma_{(c)}{}^2}}. \tag{11}$$

Suppose Y_n as the number of the newly arrived packet-in messages during the processing of the $(n + 1)$th one, its probability can be derived in (12) with its Poisson distribution assumption in the above queueing model.

$$\begin{aligned} P(Y_n = j) &= \int_0^{\infty} P(Y_n = j | T_n = x) g(x) dx \\ &= \int_0^{\infty} \frac{(\lambda_{(c)} x)^j}{j!} e^{-\lambda_{(c)} x} g(x) dx,\ j = 0, 1, 2, \cdots \end{aligned} \tag{12}$$

Suppose $a_j = P(Y_n = j) > 0$, it can be proved that $\{X_n, n \geq 1\}$ makes up a Markov chain, whose one-step transfer matrix and state transfer diagram are illustrated in (13) and Fig. 4, respectively.

$$P = \begin{pmatrix} a_0 & a_1 & a_2 & \cdots \\ a_0 & a_1 & a_2 & \cdots \\ 0 & a_0 & a_1 & \cdots \\ 0 & 0 & a_0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{13}$$

5.4. Performance parameter solution

Suppose the controller has the traffic intensity of packet-in messages $\rho_{(c)} = \lambda_{(c)}/\mu_{(c)}$, it can be verified that the Markov chain $\{X_n, n \geq 1\}$ is ergodic if $\rho_{(c)} < 1$. So there

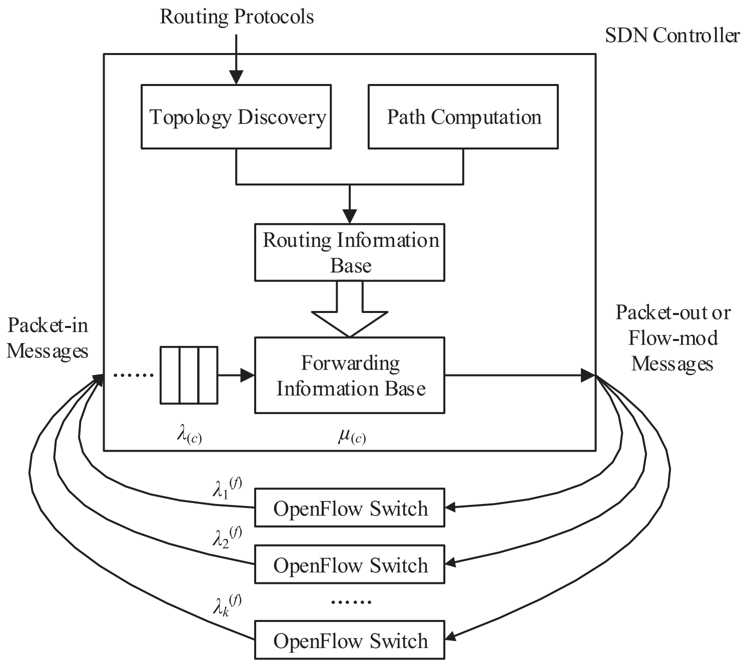


Fig. 3. The packet-in message processing of a SDN controller.

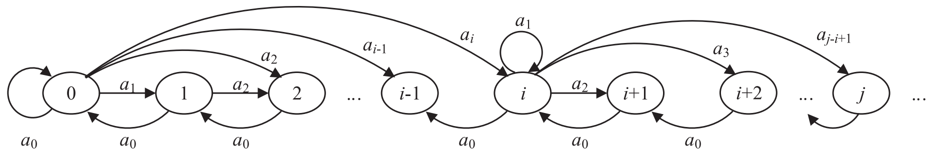


Fig. 4. The state transfer diagram of the Markov chain.

exist the stationary distribution $\{p_j, j \geq 0\}$, which satisfies the following equation also expressed as $(p_0, p_1, p_2, \ldots) = (p_0, p_1, p_2, \ldots)P$.

$$p_j = \sum_{i=0}^{\infty} p_i p_{ij}, j \geq 0. \tag{14}$$

Then, p_j is resolved with generating functions. Let $P(x)$ and $A(x)$ be the generating functions of X_n and Y_n respectively in (15) and (16).

$$P(x) = \sum_{j=0}^{\infty} p_j x^j. \tag{15}$$

$$A(x) = \sum_{j=0}^{\infty} a_j x^j. \tag{16}$$

With $P(1) = 1$, $A(1) = 1$ and $A'(1) = E(Y_n) = \rho_{(c)}$, we can derive the following generating function $P(x)$.

$$P(x) = \frac{(1 - \rho_{(c)})(1 - x)A(x)}{A(x) - x}. \tag{17}$$

Besides, we can get the result $A''(1) = {\rho_{(c)}}^2 + {\lambda_{(c)}}^2 {\sigma_{(c)}}^2$ by expressing the variance of Y_n with $A(x)$ as $D(Y_n) = A''(1) + A'(1) - [A'(1)]^2$, and directly computing the variance of Y_n as $D(Y_n) = \rho_{(c)} + {\lambda_{(c)}}^2 {\sigma_{(c)}}^2$ with $E(Y_n) = \rho_{(c)}$ and $E(Y_n^2) = \rho_{(c)} + {\rho_{(c)}}^2 + {\lambda_{(c)}}^2 \sigma_{(c)}$.

Then, we can derive the average queue length of packet-in messages in the controller in (18) from the generating function $P(x)$ by L'Hospital's rule. Subsequently, we can further get the average sojourn time of packet-in messages in the controller in (19) in terms of Little formula.

$$\overline{L_{(c)}} = E(X_n) = P'(1) = \rho_{(c)} + \frac{{\rho_{(c)}}^2 + {\lambda_{(c)}}^2 {\sigma_{(c)}}^2}{2(1 - \rho_{(c)})}. \tag{18}$$

$$\overline{W^{(c)}} = \frac{\overline{L^{(c)}}}{\lambda_{(c)}} = \frac{1}{\mu_{(c)}} + \frac{{\rho_{(c)}}^2 + {\lambda_{(c)}}^2 {\sigma_{(c)}}^2}{2\lambda_{(c)}(1 - \rho_{(c)})}. \tag{19}$$

6. Queueing model of OpenFlow networks

In OpenFlow networks, a switch maintains a queue for all arrived packets through each ingress port, and forwards them in term of its internal flow tables. If there is no entry in the tables matching the packet initiating a new flow, the switch will send a flow setup request to its SDN controller and receive the respective flow rule from the controller to guide the forwarding of packets within the flow. Meanwhile, the controller keeps a queue for all flow setup requests, i.e., packet-in messages, from its subordinate switches. With the above queueing analysis of OpenFlow switches and SDN controllers respectively

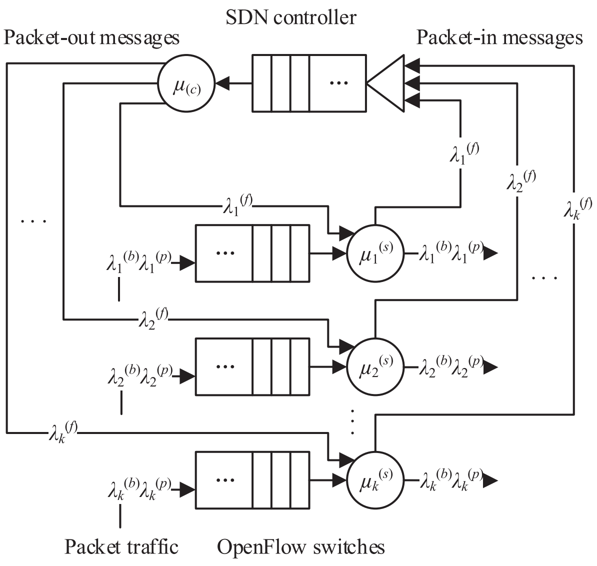


Fig. 5. The queueing model of packet forwarding in OpenFlow networks.

in Sections 4 and 5, we can model packet forwarding performance in OpenFlow networks as a queueing system in Fig. 5.

As illustrated in Fig. 5, the ith OpenFlow switch with the forwarding rate $\mu_i^{(s)}$ receives packets in batch with the rate $\lambda_i^{(b)}\lambda_i^{(p)}$. Suppose an arrived packet in the ith switch belongs to a new flow with the probability q_i, the switch sends packet-in messages with the rate $\lambda_i^{(f)}$ in (20) to its SDN controller. The controller with the processing rate $\mu_{(c)}$ receives packet-in messages with the rate $\lambda_{(c)}$ in (8) from k OpenFlow switches. The responses of packet-in messages are sent back to continue the forwarding of the corresponding packets and update the flow tables in the respective switch.

$$\lambda_i^{(f)} = q_i\lambda_i^{(b)}\lambda_i^{(p)}. \tag{20}$$

Let W_i be the forwarding time of a packet through the ith OpenFlow switch, then we can compute it in (21) by dividing the packet into two cases: directly forwarding and forwarding with the involvement of the controller. The packet forwarding time of the latter case chiefly consists of two parts, the sojourn time of the packet in the switch $W_i^{(s)}$, and the sojourn time of the respective packet-in message in the controller $W^{(c)}$.

$$W_i = \begin{cases} W_i^{(s)} & \text{with the probability } 1-q_i, \\ W_i^{(s)} + W^{(c)} & \text{with the probability } q_i. \end{cases} \tag{21}$$

As in (21), the expectations of $W_i^{(s)}$ and $W^{(c)}$ have been inferred respectively in (6) and (19). With the above results, we can calculate the average packet forwarding time through an OpenFlow switch in (22). Finally, we resolve the probability density function (PDF) of the packet forwarding time $W_i(t)$ in (23) shown in the Theorem 6.1, and further compute the cumulative distribution function (CDF) of that in (31).

$$\begin{aligned}\overline{W_i} &= E[W_i] = (1-q_i)E[W_i^{(s)}] + q_i(E[W_i^{(s)}] + E[W^{(c)}]) \\ &= \overline{W_i^{(s)}} + q_i\overline{W^{(c)}} \\ &= \frac{\lambda_i^{(p)}+1}{2(\mu_i^{(s)} - \lambda_i^{(b)}\lambda_i^{(p)})} + q_i\left(\frac{1}{\mu_{(c)}} + \frac{{\rho_{(c)}}^2 + {\lambda_{(c)}}^2{\sigma_{(c)}}^2}{2\lambda_{(c)}(1-\rho_{(c)})}\right).\end{aligned} \tag{22}$$

Theorem 6.1. *Let W_i in (21) be the forwarding time of a packet through the ith $(i = 1, 2, \dots, k)$ OpenFlow switch where $W_i^{(s)}$ and $W^{(c)}$ respectively represent the packet sojourn time in the ith switch and the controller, suppose $a_i = 1/\overline{W_i^{(s)}}$ and $a_c = 1/\overline{W^{(c)}}$, then the probability density function of the packet forwarding time W_i can be derived as (23).*

$$\begin{aligned}W_i(t) = {} & (1-q_i)a_ie^{-a_it} + \frac{q_ia_i}{\sqrt{2\pi}}e^{\frac{(a_i^2{\sigma_{(c)}}^2}{2} + \frac{a_i}{a_c} - a_it)} \\ & \times\left[\Phi\left(\frac{t - 1/a_c - a_i{\sigma_{(c)}}^2}{\sigma_{(c)}}\right)\right. \\ & \left. - \Phi\left(\frac{-1/a_c - a_i{\sigma_{(c)}}^2}{\sigma_{(c)}}\right)\right].\end{aligned} \tag{23}$$

Proof 6.1. According to the assumptions, a_i and a_c respectively denote the average processing rates of the ith switch and the controller. So, we can get the probability density functions of the packet sojourn time in the switch and the controller respectively in (24) and (25).

$$f_i(t) = a_ie^{-a_it}. \tag{24}$$

$$g(t) = \frac{1}{\sqrt{2\pi}\sigma_{(c)}}e^{-\frac{(t-1/a_c)^2}{2{\sigma_{(c)}}^2}}. \tag{25}$$

Then, we can derive the following expression from (21) by using Laplace transformation.

$$\begin{aligned}W_i(s) = E\left[e^{-sW_i}\right] &= (1-q_i)\frac{a_i}{a_i+s} \\ &+ q_i\left[\frac{a_i}{a_i+s}\times\frac{1}{\sqrt{2\pi}}e^{\left(\frac{{\sigma_{(c)}}^2s^2}{2} - \frac{s}{a_c}\right)}\Phi\left(\frac{1}{a_c\sigma_{(c)}} - \sigma_{(c)}s\right)\right].\end{aligned} \tag{26}$$

Let $\Psi(s)$ in (27) be the tail of (26). Since there are (28) and (29), we resolve the inverse Laplace transformation of $\Psi(s)$ in (30) according to the convolution theorem.

$$\Psi(s) = \frac{a_i}{a_i+s}\times\frac{1}{\sqrt{2\pi}}e^{\left(\frac{{\sigma_{(c)}}^2s^2}{2} - \frac{s}{a_c}\right)}\Phi\left(\frac{1}{a_c\sigma_{(c)}} - \sigma_{(c)}s\right) \tag{27}$$

$$L^{-1}\left(\frac{a_i}{a_i+s}\right) = a_ie^{-a_it}. \tag{28}$$

$$\begin{aligned}& L^{-1}\left[\frac{1}{\sqrt{2\pi}}e^{\left(\frac{{\sigma_{(c)}}^2s^2}{2} - \frac{s}{a_c}\right)}\Phi\left(\frac{1}{a_c\sigma_{(c)}} - \sigma_{(c)}s\right)\right]) \\ &= \frac{1}{\sqrt{2\pi}\sigma_{(c)}}e^{-\frac{(t-1/a_c)^2}{2{\sigma_{(c)}}^2}}.\end{aligned} \tag{29}$$

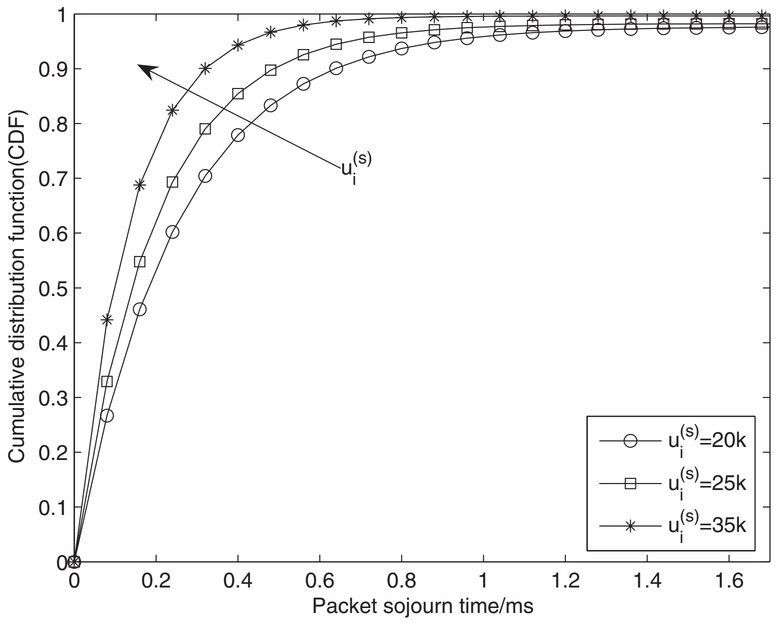


Fig. 6. The cumulative distribution function of packet sojourn time.

$$
\begin{aligned}
L^{-1}(\Psi(s)) &= \left(a_i e^{-a_i t}\right) * \left[\frac{1}{\sqrt{2\pi}\sigma_{(c)}} e^{-\frac{(t-1/a_c)^2}{2\sigma_{(c)}{}^2}}\right] \\
&= \int_0^t a_i e^{-a_i(t-\tau)} \frac{1}{\sqrt{2\pi}\sigma_{(c)}} e^{-\frac{(\tau-1/a_c)^2}{2\sigma_{(c)}{}^2}} d\tau \\
&= \frac{a_i}{\sqrt{2\pi}} e^{\frac{(a_i^2\sigma_{(c)}{}^2}{2}+\frac{a_i}{a_c}-a_i t)} \\
&\quad \left[\Phi\left(\frac{t-1/a_c-a_i\sigma_{(c)}{}^2}{\sigma_{(c)}}\right)\right. \\
&\quad \left.-\Phi\left(\frac{-1/a_c-a_i\sigma_{(c)}{}^2}{\sigma_{(c)}}\right)\right] \qquad (30)
\end{aligned}
$$

Until now, we can derive the inverse Laplace transformation of $W_i(s)$ in (26) as (23) with (28) and (30). Thus, the above theorem is proved.

$$\widetilde{W}_i(t) = P(W_i > t) = \int_t^{+\infty} W_i(t)dt. \qquad (31)$$

□

7. Performance evaluation

This section evaluates our proposed queueing model of OpenFlow networks with different performance parameters by using numerical analysis, and our controller performance model under various network scenarios by using the prevalent benchmark Cbench.

7.1. Numerical analysis

Suppose there are 128 OpenFlow switches connected to a SDN controller with the processing rate of packet-in messages $\mu_c = 16K$, and each switch with the packet forwarding rate $\mu_i^{(s)} = 32k$ receives packet traffic with the batch rate $\lambda_i^{(b)} = 256$ and the average number of packets per batch $\lambda_i^{(p)} = 8$. According to network traffic measurements, a packet at the switch belongs to a new flow with the average probability $q_i = 2^{-5}$. Then, we can calculate the arrival rate of packet-in messages at the controller from each switch as $\lambda_i^{(f)} = 64$. The above values of model parameters are taken as the benchmark of the following numerical analysis.

With the processing rate of the controller and the probability of a packet belonging to a new flow respectively set as $\mu_c = 16K$ and $q_i = 2^{-5}$, we can calculate the cumulative distribution function of packet sojourn time in Fig. 6. As shown in Fig. 6, the higher the processing rate of the controller is, the faster the cumulative probability comes close to 1. In particular, the cumulative probability increases sharply before 0.7 and grows up slowly when the packet sojourn time goes beyond 1.

Fixing the packet forwarding rate of each switch and the average probability of a packet belonging to a new flow respectively as $\mu_i^{(s)} = 32k$ and $q_i = 2^{-5}$, we can get the packet sojourn time with the increasing batch arrival rate $\lambda_i^{(b)}$ for different message processing rate of the controller μ_c in Fig. 7. As seen from Fig. 7, the packet sojourn time slowly rises up before the batch arrival rate reaches 300 per second, and sharply increases after the rate exceeds 400 per second. Furthermore, we can also see that the higher the processing rate of the controller is, the greater the number of packet batches can go through the network with identical sojourn time.

With the packet forwarding rate of each switch and the processing rate of the controller respectively configured as $\mu_i^{(s)} = 32k$ and $\mu_c = 16K$, we can reach the relationship of network throughput and packet sojourn time for different probability of a packet belonging to a new flow q_i in Fig. 8. As shown in Fig. 8, the network throughput rises up to

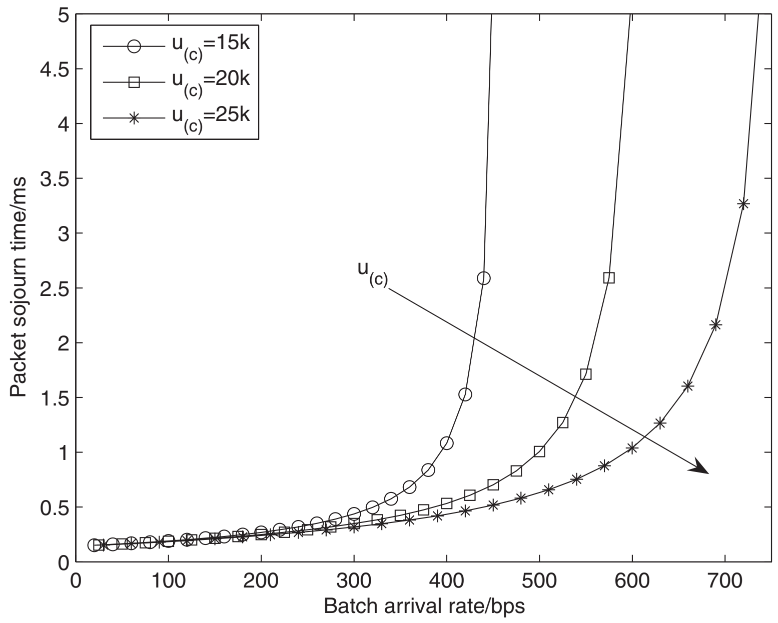


Fig. 7. The packet sojourn time with the increasing batch arrival rate.

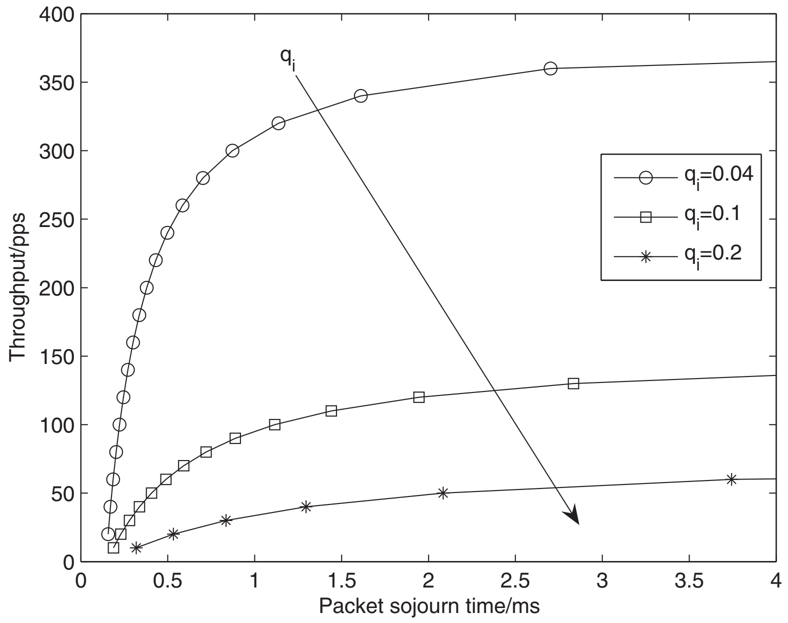


Fig. 8. The relationship of network throughput and packet sojourn time.

be constant along with the increasing packet sojourn time, and greatly relies on the probability of a packet belonging to a new flow.

Keeping the packet forwarding rate of each switch and the packet probability respectively to stay at $\mu_i^{(s)} = 32k$ and $q_i = 2^{-5}$, we can compute the packet sojourn time with the increasing number of switches for different message processing rate of the controller μ_c in Fig. 9. As seen from Fig. 9, the packet sojourn time slowly rises up at the time of the number of switches below 150, and sharply increases after the number go beyond 200. Moreover, we can also see that the higher the processing rate of the controller is, the larger the number of switches can be accommodated in the network with identical sojourn time.

7.2. Controller performance

We evaluate the controller performance with the publicly available benchmark Cbench [27], which can simulate a variety of network scenarios with different number of

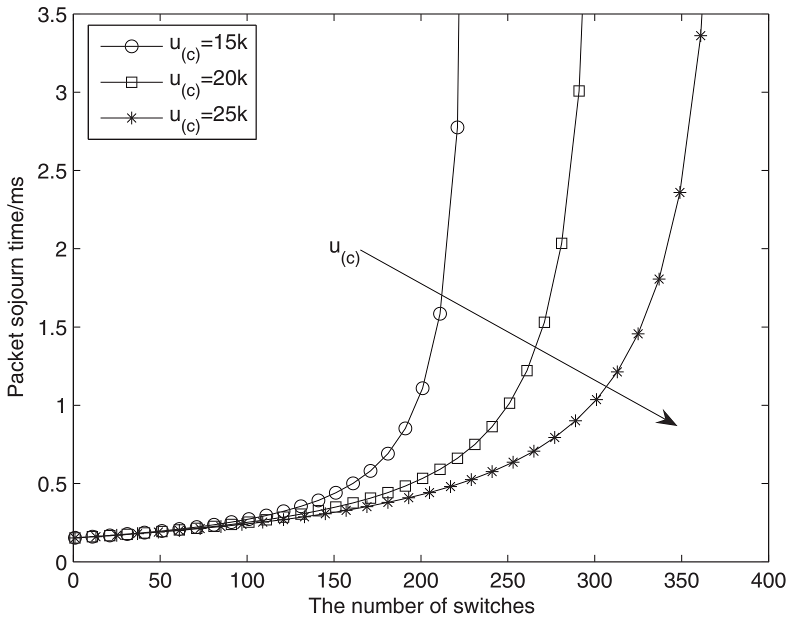


Fig. 9. The packet sojourn time with the increasing number of switches.

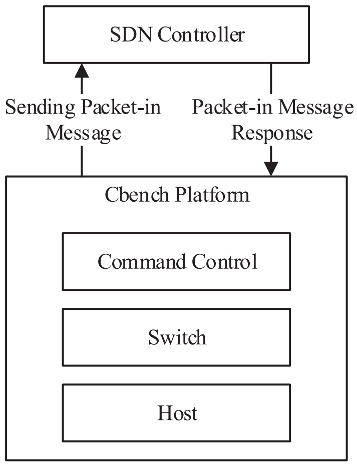


Fig. 10. The fundamental framework of the Cbench.

OpenFlow switches and PC hosts. Fig. 10 illustrates the fundamental framework of the Cbench. Under this framework, the measurements of SDN controller performance are performed with the following steps: (a) simulating a network topology by setting the amount of OpenFlow switches and hosts per switch; (b) generating a sequence of packet-in messages from each switch to the controller, and capturing their responses, i.e., packet-out messages or flow-mod messages; (c) recording the statistical information such as the amount of responses, and calculating the performance parameters of the controller such as the average sojourn time of packet-in messages.

The Cbench has two operation modes, i.e., throughput mode and latency mode. In the throughput mode, each switch continuously sends a stream of packet-in messages to the controller. The controller cannot process all the messages in real time, and results in its full-load state. Thus, this mode can be utilized to measure the processing rate of a SDN controller $\mu_{(c)}$ for each network scenario, by counting the number of response messages within a pre-set time interval. Different from the throughput mode, each switch would not send a packet-in message to the controller until it receives a response to the last message it sends in the latency mode. This implies that there are a largely fixed number of packet-in messages waiting to be processed or being processed in the controller. Hence, we can utilize this mode to measure the sojourn time of a packet-in message and estimate the arrival rate of packet-in messages $\lambda_{(c)}$ in the same way as the previous mode.

As for a given network scenario, we run the Cbench in the throughput mode with n times to get a sequence of measurement results about the processing rate of the controller $\mu_{(c)i} (i = 0, 1, 2, \ldots, n)$. Each result is calculated by dividing the number of response messages by the setting time interval. Then, we can estimate the processing rate of the controller $\mu_{(c)}$ and its standard deviation σ respectively in (32) and (33) in terms of parameter estimation in mathematical statistics. After that, the operation of the Cbench turns to the latency mode to measure the sojourn time of a packet-in message for multiple times. Each measured time is computed by dividing the counting time by the amount of received response messages. Meanwhile, the mean of their reciprocals is taken as the estimation of the arrival rate of packet-in messages $\lambda_{(c)}$. With these parameters, we can calculate the estimated sojourn time of a packet-in message for our proposed performance model $M/G/1$ and the simple one $M/M/1$ according to (19) and (34) respectively. Finally, we evaluate our proposed model by verifying whether its estimated sojourn time are more closer to all measured sojourn time of packet-in messages

Table 1
The performance parameters of both controller models.

Performance parameters	The number of switches							
	8	16	24	32	40	48	56	64
Arrival rate $\lambda_{(c)}$	8.02	7.86	7.36	6.82	6.68	6.42	6.27	6.17
Processing rate $\mu_{(c)}$	16.8	16.0	15.0	14.1	13.7	13.3	13.1	12.8
Standard deviation $\sigma_{(c)}$	0.062	0.065	0.069	0.074	0.076	0.079	0.081	0.082
Estimated delay of the $M/G/1$ model	0.116	0.125	0.133	0.140	0.145	0.149	0.151	0.154
Estimated delay of the $M/M/1$ model	0.114	0.123	0.131	0.137	0.142	0.145	0.146	0.151

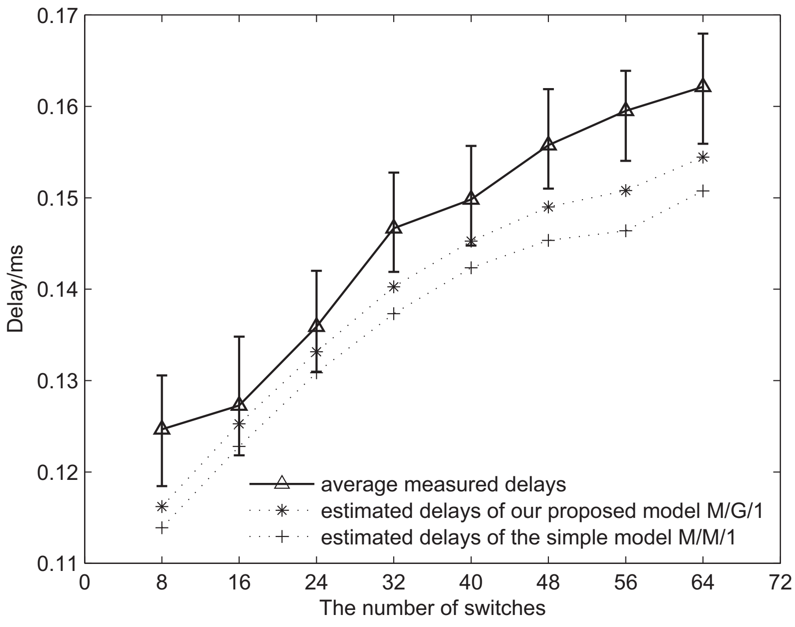


Fig. 11. The measured and estimated delays of both controller models.

than the simple one under various network scenarios.

$$\mu_{(c)} = \frac{1}{n}\sum_{i=1}^{n}\mu_{(c)i}. \tag{32}$$

$$\sigma_{(c)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\mu_{(c)i} - \mu_{(c)})^2}. \tag{33}$$

$$\overline{W} = \frac{1}{\mu_{(c)} - \lambda_{(c)}}. \tag{34}$$

With the Cbench, we compare our proposed controller model with the simple one [17] in terms of the sojourn time of packet-in messages in the well-known controller Floodlight under various network scenarios. First of all, 8 network topologies are simulated with the number of switches increasing progressively from 8 to 64 and that of hosts fixed as 1024 per switch. As for each scenario, we measure essential performance parameters including the processing rate of the controller and the sojourn time of a packet-in message. In particular, each parameter is measured for 36 times, among which the results of the first 4 times are ignored due to their unstable state at the start-up phase. Then, we estimate the expectation of other performance parameters, including the arrival rate of packet-in messages $\lambda_{(c)}$, the processing rate of the controller $\mu_{(c)}$ and its standard deviation $\sigma_{(c)}$ in Table. 1. With these parameters, we compute the estimated delays of packet-in messages for both controller models in Table. 1.

Fig. 11 illustrates all measured delays of packet-in messages with confidence intervals and the estimated ones of both controller models for the 8 network scenarios. As seen from Fig. 11, the estimated delays of our proposed model $M/G/1$ come closer to the measured delays than those of the simple one $M/M/1$ in all cases. This implies that our proposed model offers a more accurate estimation of SDN controller performance than the simple one. Besides, both estimated delays are less than almost all measured delays for each scenario in Fig. 11. This attributes to the fact that our experiments introduce the additional transmission delays between the controller and the simulated switches into the measured delays, besides of the queueing and processing delays only considered by queueing models.

8. Conclusion and future work

Understanding the performance and limitation of OpenFlow-based SDN is a prerequisite of its deployments. To achieve this goal, this paper present an analytical performance model of OpenFlow networks based on queueing theory, and resolves its average packet sojourn time through a switch. The numerical analysis on our performance model indicate that the packet forwarding performance in large-scale OpenFlow networks greatly relies on the packet-in message processing capacity of the controller. Furthermore, the experiments with the controller benchmark Cbench indicate that our controller model is a more precise approximation of SDN controller performance than the other ones.

In our future work, we will apply the queueing theory to build analytical performance models for other prevalent SDN deployments such as data center networks. Meanwhile, we will extend the analytical performance models from single SDN controller to the case of controller clusters. Moreover, we also plan to carry out the performance evaluation of the controller with our performance model to other popular controllers, such as NOX, POX, Beacon, Maestro and Ryu, which aims to provide a comprehensive understanding of SDN controller performance.

Acknowledgment

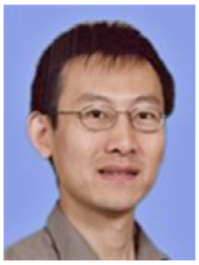
This work was supported in part by National Natural Science Foundation of China (61502056 and 61303043), Hunan Provincial Natural Science Foundation of China (2015JJ3010 and 13JJ4052), and Scientific Research Fund of Hunan Provincial Education Department (15B009).

References

[1] S. Rowshanrad, S. Namvarasl, V. Abdi, M. Hajizadeh, M. Keshtgary, A survey on SDN, the future of networking, J. Adv. Comput. Sci. Technol. 3 (2) (2014) 232–248.
[2] A. Hakiri, A. Gokhale, P. Berthou, D.C. Schmidt, T. Gayraud, Software-defined networking: Challenges and research opportunities for future internet, Comput. Netw. 75 (24) (2014) 453–471.
[3] J. Pan, S. Paul, R. Jain, A survey of the research on future internet architectures, IEEE Commun. Mag. 49 (7) (2011) 26–36.
[4] A. Lara, A. Kolasani, B. Ramamurthy, Network innovation using openflow: A survey, IEEE Commun. Surv. Tutor. 16 (1) (2014) 493–512.
[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: Enabling innovation in campus networks, ACM SIGCOMM Comput. Commun. Rev. (CCR) 38 (2) (2008) 69–74.
[6] N. Feamster, J. Rexford, E. Zegura, The road to SDN: An intellectual history of programmable networks, Networks 11 (12) (2013) 20–40.
[7] B.N. Astuto, M. Mendonca, X.N. Nguyen, K. Obraczka, T. Turletti, A survey of software-defined networking: Past, present, and future of programmable networks, IEEE Commun. Surv. Tutor. 16 (3) (2014) 1617–1634.
[8] A. Greenberg, G. Hjalmtysson, D.A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, H. Zhang, A clean slate 4D approach to network control and management, ACM SIGCOMM Comput. Commun. Rev. 35 (5) (2005) 41–54.
[9] M. Casado, M.J. Freedman, J. Pettit, J. Luo, N. McKeown, S. Shenker, Ethane: Taking control of the enterprise, ACM SIGCOMM Comput. Commun. Rev. 37 (4) (2007) 1–12.
[10] F. Benamrane, M.B. Mamoun, R. Benaini, Short: A case study of the performance of an openflow controller, in: Proceedings of the Second International Conference on Lecture Notes in Computer Science (LNCS), 2014, pp. 330–334.
[11] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Holzle, S. Stuart, A. Vahdat, B4: Experience with a globally-deployed software defined wan, ACM SIGCOMM Comput. Commun. Rev. 43 (4) (2013) 3–14.
[12] C.Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, R. Wattenhofer, Achieving high utilization with software-driven WAN, ACM SIGCOMM Comput. Commun. Rev. 43 (4) (2013) 15–26.
[13] Z. Bozakov, A. Rizk, Taming SDN controllers in heterogeneous hardware environments, in: Proceedings of the Second European Workshop on Software Defined Networks (EWSDN), 2013, pp. 50–55.
[14] S. Azodolmolky, P. Wieder, R. Yahyapour, Performance evaluation of a scalable software-defined networking deployment, in: Proceedings of the Second European Workshop on Software Defined Networks (EWSDN), 2013, pp. 68–74.
[15] S. Azodolmolky, R. Nejabati, M. Pazouki, P. Wieder, An analytical model for software defined networking: A network calculus-based approach, Proceedings of the 2013 IEEE Global Communications Conference (GLOBECOM), 2013, pp. 1397–1402.
[16] A.G. Osgouei, A.K. Koohanestani, H. Saidi, A. Fanian, Analytical performance model of virtualized SDNs using network calculus, in: Proceedings of the Twenty-third Iranian Conference on Electrical Engineering (ICEE), 2015, pp. 770–774.
[17] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, T.G. Phuoc, Modeling and performance evaluation of an openflow architecture, in: Proceedings of the Twenty-third International Teletraffic Congress (ITC), 2011, pp. 1–7.
[18] K. Mahmood, A. Chilwan, O. Osterbo, M. Jarschel, Modelling of openflow-based software-defined networks: the multiple node case, IET Netw. 4 (5) (2015) 278–284.
[19] Q. Zuo, M. Chen, P. Jiang, Delay evaluation of openflow control plane by queue model, J. Huazhong Univ. Sci. Technol. (Nat. Sci. Ed.) 8 (1) (2013) 44–49.
[20] L. Yao, P. Hong, W. Zhou, Evaluating the controller capacity in software defined networking, in: Proceedings of the Twenty-third International Conference on Computer Communication and Networks (ICCCN), 2014, pp. 1–6.
[21] H. Farhady, H.Y. Lee, A. Nakao, Software-defined networking: A survey, Comput. Netw. 81 (22) (2015) 79–95.
[22] D. Kreutz, F.M.V. Ramos, P. Verissimo, E.C. Rothenberg, S. Azodolmolky, S. Uhlig, Software-defined networking: A comprehensive survey, Proc. IEEE 103 (1) (2015) 14–76.
[23] M. Jammal, T. Singh, A. Shami, R. Asal, Y. Li, Software defined networking: State of the art and research challenges, Comput. Netw. 72 (29) (2014) 74–98.
[24] F. Hu, Q. Hao, K. Bao, A survey on software-defined network and openflow: From concept to implementation, IEEE Commun. Surv. Tutor. 16 (4) (2014) 2181–2206.
[25] A. Gelberger, N. Yemini, R. Giladi, Performance analysis of software-defined networking (SDN), in: Proceedings of the Twenty-first IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2013, pp. 389–393.
[26] F. Alencar, M. Santos, M. Santana, S. Fernandes, How software aging affects SDN: A view on the controllers, in: Proceedings of the Sixth Global Information Infrastructure and Networking Symposium (GIIS), 2014, pp. 1–6.
[27] M. Jarschel, F. Lehrieder, Z. Magyari, R. Pries, A flexible openflow–controller benchmark, in: Proceedings of the First European Workshop on Software Defined Networking (EWSDN), 2012, pp. 48–53.
[28] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, S. Shenker, NOX: Towards an operating system for networks, ACM SIGCOMM Comput. Commun. Rev. 38 (3) (2008) 105–110.
[29] D. Erickson, The beacon openflow controller, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN), 2013, pp. 13–18.
[30] Z. Cai, A.L. Cox, N.T.S. Eugene, Maestro: Balancing Fairness, Latency and Throughput in the OpenFlow Control Plane, Rice University, 2011. Technical report 10-11.
[31] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, R. Sherwood, On controller performance in software-defined networks, in: Proceedings of the Second USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE), 2012, p. 10.
[32] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, R. Smeliansky, Advanced study of SDN/OpenFlow controllers, in: Proceedings of the Ninth Central and Eastern European Software Engineering Conference in Russia, 2013, pp. 1–4.
[33] Z. Cai, Maestro: Achieving Scalability and Coordination in Centralized Network Control Plane, (Ph.d. thesis). Rice University, 2011.
[34] S.A. Shah, J. Faiz, M. Farooq, A. Shafi, S.A. Mehdi, An architectural evaluation of SDN controllers, in: Proceedings of the 2013 IEEE International Conference on Communications (ICC), 2013, pp. 3504–3508.
[35] R. Jain, S.A. Routhier, Packet trains: measurements and a new model for computer network traffic, IEEE J. Sel. Areas Commun. 4 (6) (1986) 986–995.

[36] B.D. Choi, D.I. Choi, Y. Lee, D.K. Sung, Priority queueing system with fixed-length packet-train arrivals, IEE Proc. Commun. 145 (5) (1998) 331–336.
[37] V. Paxson, S. Floyd, Wide area traffic: The failure of Poisson modeling, IEEE/ACM Trans. Netw. (ToN) 3 (3) (1995) 226–244.
[38] H. Okamura, T. Dohi, K.S. Trivedi, Markovian arrival process parameter estimation with group data, IEEE/ACM Trans. Netw. 17 (4) (2009) 1326–1339.
[39] C.W. Huang, S. Sukittanon, J. Ritcey, A. Chindapol, J.N. Hwang, An embedded packet train and adaptive FEC scheme for VoIP over wired/wireless IP networks, in: Proceedings of the 2006 IEEE International Conference on Acoustics, Speech and Signal Processing (ASSP), vol. 5, 2006, pp. 429–432.
[40] P. Zalan, S. Molnar, T. Elteto, Jitter analysis of multimedia streaming traffic assuming batch arrivals, in: Proceedings of the 2008 Euro-NGI Conference on Next Generation Internet Networks (NGI), 2008, pp. 268–275.
[41] C. Williamson, Internet traffic measurement, Internet Comput. 5 (6) (2001) 70–74.
[42] J. Yang, W.C. Li, Y.Y. Qiao, Z.M. Fadlullah, N. Kato, Characterizing and modeling of large-scale traffic in mobile network, in: Proceeding of the 2015 IEEE Wireless Communications and Networking Conference (WCNC), 2015, pp. 801–806.
[43] Z. Liang, K. Xu, J. Wu, A novel model to analyze the performance of routing lookup algorithms, in: Proceeding of the 2003 IEEE International Conference on Communication Technology (ICCT), 2003, pp. 508–513.

Bing Xiong received his Ph.D. in 2009 by master-doctorate program from the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), China, and B.Sc. in 2004 from Hubei Normal University, China. He is currently a lecturer in the School of Computer and Communication Engineering, Changsha University of Science and Technology, China. His main research interests include software defined networks, future internet architecture, network security.



Kun Yang received his Ph.D. from the Department of Electronic & Electrical Engineering of University College London (UCL), UK, and M.Sc. and B.Sc. from the Computer Science Department of Jilin University, China. He is currently a full Professor in the School of Computer Science & Electronic Engineering, University of Essex, UK. Before joining in University of Essex at 2003, he worked at UCL on several European Union (EU) research projects for several years. His main research interests include heterogeneous wireless networks, fixed mobile convergence, pervasive service engineering, future Internet technology and network virtualization, cloud computing. He manages research projects funded by various sources such as UK EPSRC, EU FP7 and industries. He has published 60+ journal papers. He serves on the editorial boards of both IEEE and non-IEEE journals. He is a Senior Member of IEEE and a Fellow of IET.

Jinyuan Zhao received her M.S. and B.S. in 2007 and 2004, respectively, from Central China Normal University and Hubei Normal University. She is currently a lecturer in the Department of Computer and Communication, Hunan Institute of Engineering, China, and also a Ph.D. candidate of software engineering in the School of Software, Central South University (CSU), China. Her main research interests include cloud computing and future networks.

Wei Li received his B.S. in 2013 from Heilongjiang Bayi Agricultural University. He is currently a master candidate in the School of Computer and Communication Engineering, Changsha University Of Science and Technology, China. His main research interests include software defined networks and network performance evaluation.

Keqin Li received B.S. degree in computer science from Tsinghua University, Beijing, China, in 1985, and Ph.D. degree in computer science from the University of Houston, Houston, Texas, USA, in 1990. He was a full professor (1999–2009), and has been a SUNY distinguished professor of computer science since 2009 in State University of New York at New Paltz. He was the acting chair of Department of Computer Science during Spring 2004. Since November 2011, he has been an Intellectual Ventures endowed visiting chair professor at the National Laboratory for Information Science and Technology, Tsinghua University, Beijing, China. He has been a IEEE Fellow since January 2015.