Cutting Long-Tail Latency of Routing Response in Software Defined Networks

Junjie Xie, Deke Guo, Xiaozhou Li, Yulong Shen, and Xiaohong Jiang

***Abstract*— To enable the network softwarization, network function virtualization (NFV) and software defined networking (SDN) are integrated to jointly manage and utilize the network resource and virtualized network functions (VNFs). For a network flow resulting from any NFV application, an associated switch would send a routing request to the controller in SDN. The controller then generates and configures a routing path to dynamically steer the flow across appropriate VNFs or service function chains. This process, however, exhibits a skew distribution of response latency with a long tail. Cutting the long-tail latency of response is critical to enable the network softwarization, yet difficult to achieve due to many factors, such as the limited capacities and the load imbalance among controllers. In this paper, we reveal that such flow requests still experience the long-tail response latency, even using the up-to-date controller-to-switch assignment mechanism. To tackle this essential problem, we first propose a light-weight and load-aware switch-to-controller selection scheme to cut the long-tail response latency under the simple scenario of homogeneous controllers, and then design a general delay-aware switch-to-controller selection scheme to fundamentally cut the long-tail response latency for the more complicated heterogeneous controller scenario with performance fluctuations. The comprehensive evaluations indicate that our two new switch-to-controller selection schemes can significantly reduce the long-tail latency and provide higher system throughput.**

***Index Terms*— Network softwarization, software defined networks, controller selection, long-tail latency.**

Manuscript received September 30, 2017; revised February 5, 2018; accepted February 27, 2018. This work was supported in part by the National Natural Science Foundation for Outstanding Excellent Young Scholars of China under Grant 61422214, in part by the National Natural Science Foundation of China under Grant 61772544 and Grant U1536202, in part by the National Basic Research Program (973 program) under Grant 2014CB347800, in part by the Hunan Provincial Natural Science Fund for Distinguished Young Scholars under Grant 2016JJ1002, and in part by the Guangxi Cooperative Innovation Center of cloud computing and Big Data under Grant YD16507 and Grant YD17X11. *(Corresponding authors: Deke Guo; Yulong Shen.)*

J. Xie is with the Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha 410073, China (e-mail: xiejunjie06@gmail.com).

D. Guo is with the College of System Engineering, National University of Defense Technology, Changsha 410073, China. He is also with the School of Computer Science and Technology, Tianjin University, Tianjin 300072, China (e-mail: guodeke@gmail.com).

X. Li is with the Department of Computer Science, Princeton University, Princeton, NJ 08544 USA (e-mail: xl@cs.princeton.edu).

Y. Shen is with the School of Computer Science and Technology, Xidian University, Xi'an 710071, China (e-mail: ylshen@mail.xidian.edu.cn).

X. Jiang is with the School of Systems Information Science, Future University Hakodate, Hokkaido 041-8655, Japan (e-mail: jiang@fun.ac.jp).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2018.2815358

I. Introduction

NETWORK softwarization is a transformation trend for designing, implementing, and managing the next generation networks. It exploits the benefits of software to enable the redesign of network and service architectures, optimize the expenditure and operational costs, and bring added values. The key enablers consist of the network function virtualization (NFV), software-defined networking (SDN) and cloud computing, etc [1]. Moreover, 5G systems will also rely on these technologies to attain system's flexibility and true elasticity [2], [3]. Network functions (NFs) are crucial for improving network security by examining and modifying network flows using special-purpose hardware. Recently, NFV has been proposed to execute virtual network functions (VNFs) on generic compute resources [4], such as commodity servers and VMs. Normally, a flow goes through specific VNFs in a particular order to meet its required processing, following the service function chain (SFC) [5], [6] along a routing path.

Additionally, SDN offers the freedom to refactor the control plane and flexibly enables the network softwarization [7]. More precisely, NFV and SDN can jointly manage the network resource and VNFs, and dynamically steer network flows across appropriate VNFs or SFCs. SDN centralizes the network control plane to a programmable software component, i.e., a controller running on a generic server, such as NOX [8]. The controller maintains a global network view and optimizes the forwarding decisions of network flows. For a flow from any NFV application, an associated switch would send a routing request to the controller. It is the controller that generates and configures a routing path to a specific VNF instance or to traverse a SFC on demand. The above process between a pair of switch and controller brings the response latency. Many factors would skew the tail of the latency distribution. For example, a single controller that lacks sufficient capacity to tackle received routing requests quickly and inevitably becomes a performance bottleneck [9]. Thus, such routing requests experience long-tail latency of response, as evaluated in Section II. Cutting the long-tail latency of routing response is critical to enable the network softwarization, yet difficult to achieve due to many factors.

To improve the scalability of SDN, the distributed control plane consisting of multiple controllers has been proposed recently [10], such as ONOS and OpenDaylight. To reduce the long-tail latency of response, they resort to the controller-to-switch assignment mechanism. That is, the control plane

IEEE Proof

0733-8716 © 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

proactively assigns a controller to each switch such that each controller manages the same amount of switches. In reality, the quantities of routing requests coming from switches per unit time are different and dynamic. Consequently, controllers still differ in the amount of received routing requests per time unit. This load imbalance among controllers leads to the long-tail latency of response. Additionally, the controller-to-switch assignment requires coordination among controllers, which further aggravates the loads of controllers.

In this paper, to cut the long-tail latency of response and lighten the load of controllers, we propose conducting the selection of controllers at the side of each switch instead of the controller-to-switch assignment. This means that each switch actively chooses one controller from multiple available ones, decoupling the static binding between switches and controllers. More precisely, each switch prefers to adaptively select the controller with low response latency for routing requests. This would move the partial intelligences of the network to switches and efficiently reduces the loads of controllers.

Despite those potential benefits, the selection of controllers still faces many challenges. First, the switches need to probe the state of controllers via the secure channel between them. The secure channel is one kind of rare resource and affects the performance of the whole network. To save the bandwidth of the secure channel, the selection process of controllers should be light-weight and use a few of the feedbacks from the controllers. Second, the selection scheme needs to be scalable, irrespective of the network size and the number of controllers. Third, the selection scheme needs to accommodate the bursty and skew routing requests from switches. Last, the selection scheme should adapt to the heterogeneous controllers and the performance fluctuation across controllers.

To tackle such challenges, we design a *load-aware* selection scheme of controllers, which is simple but effective to achieve the load balance among controllers. The load of a controller refers to the number of routing requests waited to be processed. The basic idea of our scheme is that each switch sends routing requests to the controller with the lowest load. In this way, all controllers process the similar number of routing requests per time slot. This is very helpful to cut the long-tail latency of routing response, when all controllers have the same processing capabilities. This method alone, however, is insufficient to deal with more general settings of heterogenous controllers and the performance fluctuation. Those controllers with lower processing capabilities still incur the long-tail latency of response for routing requests, when all controllers achieve the load balance. For this reason, we further present a general *delay-aware* selection scheme of controllers to fundamentally cut the long-tail latency of routing response.

Our *delay-aware* selection scheme includes two key components. The first one is the controller selection model of each switch, which uses simple and inexpensive probing feedbacks from a few controllers. It is still effective if each switch just randomly probes two controllers and sends upcoming routing requests to the controller with the shorter response delay. This model is scalable and light-weight since it is not affected by the network scale and the number of controllers. The second component is the queue management mechanism of each controller. It could estimate the response delay of a routing request and hence improve the performance predictability of controllers. The evaluation results reveal that our *delay-aware* selection scheme can efficiently reduce the long-tail latency of routing responses and improve the system throughput.

In summary, the major contributions of this paper are as follows.

1) We reveal that routing requests experience the long-tail response latency, even using the up-to-date controller-to-switch assignment mechanism in SDN. Therefore, we propose an adaptive selection mechanism of controllers for switches to cut the long-tail latency.
2) We first design an efficient *load-aware* selection method of homogeneous controllers for each switch. For more general scenarios, we further propose a general *delay-aware* selection method, which is adaptive to the bursty routing requests, the heterogenous controllers and the performance fluctuations. Our two methods are light-weight as they limit the additional overhead caused by probing two controllers.
3) We further develop a queue management mechanism for each controller, which can efficiently manage the queue length and estimate the response delay of routing requests. The evaluation results reveal that our controller selection methods can accommodate system environment variations and efficiently reduce long-tail latency of routing response.

The paper is organized as follows. In Section II, we present the observation of long-tail latency of routing response. Section III depicts the framework of our controller selection mechanism and the load-aware selection scheme of controllers. We present the delay-aware selection scheme of controllers in Section IV. We conduct massive experiments to evaluate the performance of our controller selection schemes under various system environment in Section V. Section VI introduces the related work. In Section VII, we conclude this paper.

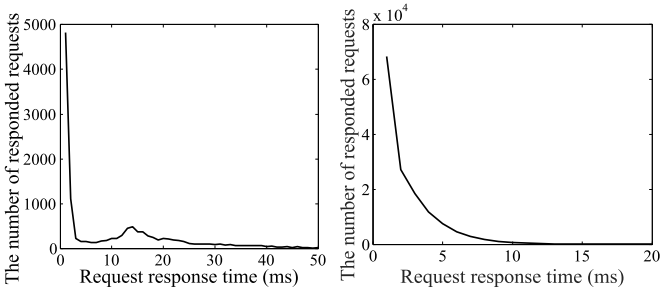
II. Long Tail of Response Latencies

In a SDN, when a switch receives a new flow, the switch sends a routing request to its controller. The controller then computes a route for the flow and inserts flow rules to related switches in the route. Thus, the new flow would be forwarded according to the flow rules in switches [11], [12]. Such an interaction between the switch and the controller causes the response latency. For a routing request, the latency of routing response denotes the time interval from sending the routing request to receiving the flow rules generated by the controller.

A. Long-Tail Observations of Response Latencies

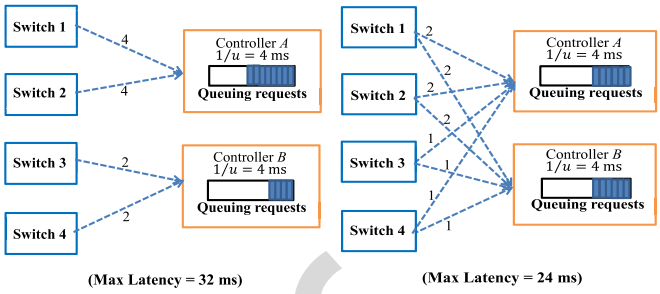
Fig. 1(a) plots an observation about the long-tail distribution under a single instance of ONOS controller. We build a SDN testbed with one controller, running in a virtual machine with 2 CPU cores and 2G RAM. Note that the testbed forms a typical Fat-tree datacenter topology [13]. We record the response latencies of 12,000 routing requests. As shown in Fig. 1(a), the response latencies of 50% of routing requests are lower than 5ms, and 90% of routing requests are

IEEE Proof



(a) A single controller with 12,000 routing requests. (b) Multiple controllers with 150,000 routing requests.

Fig. 1. Long-tail distributions of routing responses under a single controller as well as multiple controllers.



(a) Quantity-based controller allocation. (b) Load-aware controller selection.

Fig. 2. The performances of quantity-based controller allocation and load-aware controller selection.

served within 30ms. However, there still exist some routing requests whose response latencies are more than 50ms. That is, the response latencies exhibit a long-tail distribution.

Furthermore, we observe the response latencies of routing requests under multiple controllers [14]. We employ 300 switches and 40 controllers where each controller manages 7 or 8 switches. Each switch generates routing requests according to a Poisson arrival process with $\lambda = 0.5$ during 1ms. The processing time of each request in each controller is drawn from an exponential distribution where $\mu^{-1} = 2$ms. Each controller can process 10 requests in parallel. We run the system 1000ms and record the response latencies of routing requests. The number of arrival routing requests is about $150,000$. In Fig. 1(b), 89% of routing requests can be served in 5ms, and the response latencies of 96% of routing requests are lower than 10ms. However, some response latencies are still larger than 20ms. That is, Fig. 1(b) shows that there still exists a long-tail distribution under multiple ONOS controllers.

B. Analysis About the Long-Tail Latencies

Fig. 1(a) results from that the network only employs one controller. Due to the limited capability of the single controller, a large amount of requests have to queue in the controller. Therefore, there is a long-tail latency of routig response caused by the long queueing delays. In Fig. 1(b), the number of routing requests that each switch generates is different, even though they obey the same Poisson distribution. The skew-flow requests will make that some controllers are overload, but other controllers would be underutilized. As a consequence, there will be a long-tail latency. We illustrate the problem in Fig. 2. Two controllers are assigned to four switches and have the same processing time of 4ms. Assume $switch_1$ and $switch_2$ receive 4 requests each and that $switch_3$ and $switch_4$ receive 2 requests each. The requests received by $switch_1$ and $switch_2$ can only be processed by $controller_1$, which is assigned to manage them. This leads to a maximum latency of 32ms, but a load-aware selection obtains a maximum latency of 24ms. Fig. 2 shows that a quantity-based assignment strategy leads to long-tail latencies because it fails to accommodate the skew-flow requests.

Quantity-based allocation strategy is commonly employed by many controllers to balance the loads of controllers, such as ONOS [14]. That is a controller-to-switch assignment mechanism, which is abbreviated as the assignment mechanism of controllers. In this case, controllers coordinate to manage switches, and each controller manages an approximately equal quantity of switches. When deploying multiple instances of ONOS in a SDN, the bursty flows from a switch are sent to the same controller. Consequentially, a large number of requests have to queue in the controller. These queueing requests tend to incur long response latencies. However, those controllers, which do not receive bursty routing requests, may even be underloaded. In conclusion, the assignment mechanism fails to efficiently reduce the tail latencies of responses.

III. Framework of Controller Selection Mechanism

To overcome the drawback of assignment mechanism, we design a switch-to-controller selection scheme, which is abbreviated as the selection scheme of controllers. The selection scheme moves partial intelligences of the network to switches and relieves the loads of controllers. Meanwhile, the selection scheme can efficiently reduce tail latencies of responses.

A. Overview of Controller Selection Mechanism

For existing designs of control plane, the assignment mechanism of controllers is a static binding between switches and controllers, which fails to deal with the bursty and skew routing requests, and further incurs long response delays. To reduce the tail latencies of responses, routing requests from the same switch need to be processed by appropriate controllers. This means that the static binding between switches and controllers needs to be decoupled. A better mechanism is to enable switches to select controllers for routing requests.

For the assignment mechanism, the load balance among controllers means that each controller manages the same amount of switches. The load is denoted by the number of switches. Moreover, the load balance among controllers requires the coordination of controllers. The coordination will further aggravate the computing and communication overhead. However, in this paper, the selection scheme of controllers achieves the mapping between switches and controllers through switches conduct simple and actively probing.

IEEE Proof

Therefore, the selection scheme relieves the overhead of coordination and assignment in controllers.

In SDN, the controller periodically probes the switch state, and switches would respond to those probes and forward any switch notifications (e.g., link failures or recovery) to all live controllers [15]. When a switch suffers a failure scenario, it fails to respond to the probe. After that, the controller promptly updates its local topology state and replicates that update to all other controllers in the cluster. In addition, there have been lots of researches on how to achieve the consistency among distributed controllers [15]. These techniques are complementary to our approach.

We design the controller selection mechanism keeping in mind these four goals:

1) Light-weight: A light-weight probing method is needed to save the bandwidth of the secure channel. The probing for the controller selection uses the secure channel, which is the communication channel between the control plane and the data plane. The bandwidth of the channel can affect the performance of the whole network.
2) Scalable: The selection scheme of controllers should be irrelevant to the increasing number of controllers. The expansion of network size is common. The method of probing controller should accommodate the increase of deployed controllers and avoid to incur the communication overhead and the computing overhead.
3) Burst-immunity: The selection scheme should be burst-immune. There are bursty and skew-flow requests, which can lead to long response delays. To shorten the tail latency of responses, the selection scheme needs to accommodate the bursty and skew-flow requests.
4) Adaptive: The selection scheme of controllers should be adaptive. The capabilities of controllers may be heterogeneous and time-varying. To deal with the general situation, the selection scheme must cope and quickly react to heterogeneous and time-varying processing capabilities across controllers.

B. Load-Aware Selection Scheme of Controllers

Accommodating skew-flow requests across controllers necessitates a selection strategy of controllers. The strategy can make switches select faster controllers for routing requests. The controller with fewer unfinished requests can respond to routing requests faster when controllers have the same processing capability. In this paper, we first present a load-aware selection scheme. To realize this framework, the selection strategy needs to take into account the loads across multiple controllers in the network. The load means the number of unfinished requests. Our load-aware selection scheme is to select a controller with the fewest unfinished requests for newly generated requests.

Under the load-aware selection scheme of controllers, the switch needs to send a probing request to each controller after a switch receives a new flow. When the controller receives the probing request, it will return the number of unfinished requests to the switch. After the switch receives all probing results, it then sends the new routing request to the controller with the lightest load since that controller can respond the routing request fastest. The load-aware selection scheme aims to reduce tail latencies of responses by selecting the controller with the lightest load.

In Fig. 2, the load-aware selection scheme will work as follows. When a switch receives a new flow, it first probes the loads of controllers A and B. Based on the loads of controllers A and B, the switch sends the routing request to the controller with lightest load. This scheme can balance the loads of controllers A and B and can achieve better selection. When the controller finishes processing the routing request, it inserts the flow rules to related switches. Lastly, those switches will deal with the flow according to the actions of matched flow rules. In addition, the new arrival routing requests need to queue in controllers, when controllers are busy. However, the infinite length of queue will incur infinite response delays of routing requests. To cut the tail latency of response, it is necessary to ensure that the length of queue is finite in each controller.

C. The Condition to Finite Queue Length

We give the condition to achieve that the expected number of requests in per controller remains finite for all time. Consider the following model: requests arrive as a Poisson stream of rate λ at each switch. Requests are processed according to the first-in first-out (FIFO) protocol by controllers. The processing time for a request is exponentially distributed with mean μ. When there are m switches and n controllers in a network, requests arrive as a Poisson stream of rate $\frac{\lambda m}{n}$ at each controller. We obtain the following theorem. Note that $\frac{\lambda m}{n} < \mu$, and then the system will be stable, which means that the expected number of requests per controller remains finite in equilibrium. Theorem 1 shows that the system is stable for every $\frac{\lambda m}{\mu n} < 1$; that is, the expected number of requests in each controller remains finite for all time.

Theorem 1: The system is stable for every $\frac{\lambda m}{\mu n} < 1$; that is, the expected number of requests in each controller remains finite for all time.

Proof: When we treat all controllers as a whole and all switches as a whole, then the system can be seen as a M/M/1 system with Poisson arrival rate λm and average service rate μn. Let $P_k(t)$ denote the probability of that there are k requests in the whole system in time t. Accordingly, $P_{k+1}(t)$ denotes the probability where $k+1$ requests exist in the system in time t, and $P_k(t+\triangle t)$ denotes the probability of that there are k requests in the whole system in time $t+\triangle t$. Now we consider the evolution of the system. In the time $[t, t+\triangle t]$, the process of evolution has some attributes just as follows:

- One request comes with the probability $\lambda m \triangle t$, and the probability of no request comes is $1-\lambda m \triangle t$.
- One request departures with the probability $\mu n \triangle t$, and the probability of no request departure is $1-\mu n \triangle t$.
- The situation of more than one request comes and departures in $\triangle t$ is a small probability event, and it can be ignored.

There are 4 types of evolution process, and we list them in Table I. Take type B for an example. Since k is the number

IEEE Proof

TABLE I
FOUR TYPES OF EVOLUTION PROCESS

Type	state in time t	come	departure	state in time $(t+\triangle t)$
(A)	k	0	0	k
(B)	$k+1$	0	1	k
(C)	$k-1$	1	0	k
(D)	k	1	1	k

of requests in the whole system, so $k+1$ means that there are $k+1$ requests in the system in time t. After that, when a request departs during $\triangle t$, there would be k requests in the system in time $t+\triangle t$. Accordingly, we can get the possibility of type A is $P_k(t)(1-\lambda m\triangle t)(1-\mu n\triangle t)$, possibility of type B is $P_{k+1}(t)(1-\lambda m\triangle t)\mu n\triangle t$, possibility of type C is $P_{k-1}(t)\lambda m\triangle t(1-\mu n\triangle t)$ and possibility of type D is $P_k(t)\lambda m\triangle t\mu n\triangle t$. Then, $P_k(t+\triangle t)$ should be the sum of all 4 types, shown in Equation (1).

$$P_k(t+\triangle t) = P_k(t)(1-\lambda m\triangle t-\mu n\triangle t) + P_{k+1}(t)\mu n\triangle t + P_{k-1}(t)\lambda m\triangle t + o(\triangle t) \quad (1)$$

And let $\triangle t \to 0$, we can get a differential equation, shown in Equation (2).

$$\frac{dP_k(t)}{dt} = \lambda m P_{k-1}(t) + \mu n P_{k+1}(t) - (\lambda m+\mu n)P_k(t) \quad k=1,2,\ldots \quad (2)$$

Noted that if $k=0$, there will exist only type A and type B, shown in Equations (3) and (4).

$$P_0(t+\triangle t) = P_0(t)(1-\lambda m\triangle t) + P_1(t)(1-\lambda m\triangle t)\mu n\triangle t \quad (3)$$

$$\frac{dP_0(t)}{dt} = -\lambda m P_0(t) + \mu n P_1(t) \quad (4)$$

We just have interest in the equilibrium point, and the derivative is 0 in fixed point. Thus, we get Equation (5).

$$\begin{cases} -\lambda m P_0 + \mu n P_1 = 0 \\ \lambda m P_{k-1} + \mu n P_{k+1} - (\lambda m+\mu n)P_k = 0 \quad k \geq 1 \end{cases} \quad (5)$$

Resolve equation (5), we can get $P_k = (\lambda m/\mu n)^k P_0$. If $\frac{\lambda m}{\mu n} < 1$, then the sequence P_k will be decrease. And we know probability is non-negative, that means $P_k \geq 0$. If a sequence is bounded and monotone, it converges [16]. So there exist K, when $k > K$, $P_k = 0$. Then the expected total number of requests in all controllers remains finite. ■

Theorem 1 shows that the expected total number of requests in each controller remains finite, when $\frac{\lambda m}{\mu n} < 1$. Therefore, to achieve the finite queue length, it is essential to ensure that $\frac{\lambda m}{\mu n} < 1$. When the network size m increases, if $\frac{\lambda m}{\mu n} \geq 1$, it is necessary to increase the number of deployed controllers n, otherwise, the queue length of some controllers will be infinite, and that will incur infinite response delays. Another method to limit the queue length of controllers is to drop some routing requests, which can limit the value of λ and achieve $\frac{\lambda m}{\mu n} < 1$.

D. Limitations of Load-Aware Selection Scheme

Fig. 3 shows the Cumulative Distribution Function (CDF) of response latencies of routing requests under different schemes.

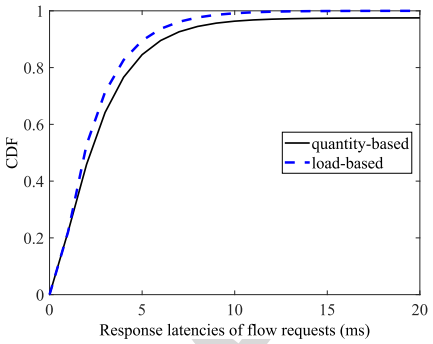


Fig. 3. Response latencies of routing requests under different schemes.

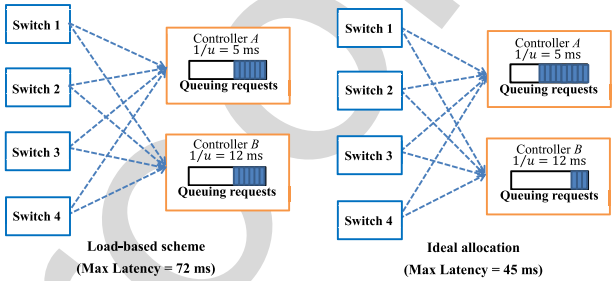


Fig. 4. Distinct selection schemes incur different response latencies.

In Fig. 3, the response latencies of 94% of routing requests are lower than 5ms after adopting the load-aware selection scheme. However, for the quantity-based assignment, only 86% of routing requests can be responded in 5ms. In Fig. 3, all routing requests can be responded in 10ms under the load-aware scheme. Therefore, Fig. 3 indicates that our load-aware selection scheme can reduce the tail latency of responses than the prior quantity-based allocation method when controllers are homogeneous and exhibit the same processing capabilities. However, we can see that both curves (Quantity-based and Load-aware) are very close to each other, which means that the load-aware selection strategy narrowly reduce the long-tail latency. Furthermore, the load-aware selection scheme faces three challenges. First, controllers are heterogeneous. Second, the processing capabilities of controllers are dynamically changing. Third, the cost of probing is too huge. Therefore, the load-aware selection scheme is still insufficient to completely cut the tail latency.

Fig. 4 plots an illustrative example of the limitation. For two controllers, the processing time per request in controller A and controller B are 5ms and 12ms, respectively. Assume all four switches receive a burst of 3 requests each. Each request needs to be forwarded to a single controller. If every switch selects a controller using the load-aware scheme, it will result in each controller receiving an equal share of the requests. This leads to a maximum latency of 72ms, whereas an ideal selection in this case obtains a maximum latency of 45ms. We note that the load-aware scheme will prefer faster controllers over time, but purely relies on the load information. Therefore, when controllers are heterogeneous, the load-aware selection scheme can not efficiently shorten the tail latency.

Controllers are commonly heterogeneous for primarily three reasons. First, the hardware is heterogeneous. Controllers run

IEEE Proof

in commercial servers. These servers can be heterogeneous due to different hardware configurations, such as CPU and memory. Second, the software is heterogenous. There are multiple different controllers developed by different organizations [10], such as NOX, Beacon, Floodlight, Ryu, ONOS and OpenDaylight, etc. Those controllers themselves have different performances. Third, the function is heterogeneous. There are some management applications running in controllers for achieving different functions [12], [17], and these applications will consume some resources of controllers. Consequentially, controllers have different remaining capabilities for processing routing requests, even if the controllers run in servers with the same setting. In this case, queueing routing requests in controllers with low processing capabilities will lead to long response latencies.

Additionally, the load-aware scheme probes the loads of all controllers, and then selects the controller that has the lightest load. However, this probing will incur the overhead of communication and aggravate the loads of controllers when there are a large number of controllers in a large-scale network. For example, there are m switches and n controllers in a network. Suppose that each switch receives λ routing requests in 1ms. There are $2\lambda \times m \times n$ times communications between switches and controllers during 1ms. Meanwhile, each controller needs to evaluate its own load λm times in 1ms. The cost of probing is too huge for the load-aware selection scheme.

The load-aware controller selection scheme can only reduce tail latencies of responses under the homogeneous controllers. However, the network environment is time-varying in real situations, not only in the processing capabilities of controllers but also in the number of routing requests from switches. We further propose a delay-aware selection scheme of controllers, which can adapt to the variations of the network environment.

IV. Delay-Aware Selection Scheme of Controllers

We design the delay-aware selection scheme of controllers while keeping the design goals of controller selection mechanism in mind. We first show an overview of the delay-aware selection scheme. Then, we present two major components of the delay-aware selection scheme, the selection models of controllers and the queue management mechanism of controller.

A. Overview of Delay-Aware Selection Scheme

To address those problems faced by the load-aware selection scheme, we further design the delay-aware selection scheme of controllers, which is adaptive to the heterogeneous controllers as well as to the dynamic behaviours of flows. The delay-aware selection scheme needs to probe the response delays of controllers for routing requests and send the routing requests to the controller with the smallest response delay. The latency of routing response denotes the time interval from sending the routing request to receiving the flow rules generated by the controller and is composed of the queueing delay and the processing delay, as shown in Definition 1. The response delay is an approximate evaluation of response time. Through probing response delays, the delay-aware selection scheme can accommodate the heterogeneous controllers, while fewer routing requests will be sent to the controllers with low processing capabilities.

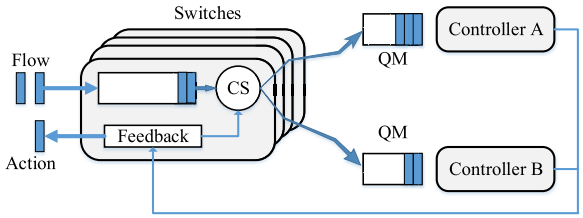


Fig. 5. Overview of controller selection scheme. CS: Controller Selection scheduler, QM: Queue Management of controller.

Definition 1: The latency of routing response denotes the time interval from sending the routing request to receiving the flow rules generated by the controller.

Furthermore, the capabilities of controllers are time-varying. With the development of SDN, there are more and more applications running in controllers. When switches send vast requests to the controller that has fast capability of response at before, a large number of requests have to queue in controllers if the capabilities of controllers decrease due to other applications' overconsumption of resources.

Our delay-aware selection scheme includes two major components, controller selection (CS) and queue management (QM). Recall the design goal of the selection scheme in Section III-A, CS can achieve that the selection scheme is light-weight, scalable and burst-immune, and QM achieves the goal of adaptivity. First, we design a selection scheme of controllers, which can select the controller based on a little feedback from the controllers, and thus, is light-weight. Second, instead of probing all controllers, the switch randomly probes d controllers where $d \geq 1$. The probing is scalable and independent of the network size. Third, the selection scheme can make switches conduct once controller selection for each flow or a batch of flows. It can make those requests be processed by different controllers, and thus can avoid the influence of the bursty and skew-flow requests. Last, through estimating response delays of routing requests, switches can send routing requests to the controller that has the smallest response delay. Based on this estimation, the selection of controllers can be adapted to the heterogeneous and time-varying processing capabilities.

Fig. 5 depicts the framework of the adaptive switch-to-controller selection scheme. When a request is issued at a switch, the switch will work based on Algorithm 1. The switch randomly probes d controllers, where $d \geq 1$. The d controllers then estimate their response delays for the routing request based on Algorithm 2 and return the response delays ψ to the switch. If ψ_i of controller i exceeds the max response delay RD_{max} limit, then controller i will return the response delay $\psi_i = -1$. When the switch receives response delays of d controllers, it will select the controller that has the smallest response delay. If all response delays are lower than 0, the switch will reselect a controller whose queue length does

IEEE Proof

Algorithm 1 The Selection of Controllers

Require: Controller set C, d.
randomly probe d controllers from C;
send estimating request to the d controllers;
$\psi \leftarrow$ response delays of d controllers;
if there exists $\psi_x \geq 0$ **then**
  $id \leftarrow arg\ min\{\psi_x \geq 0\}$;
else
  for $i = 1$ to C.length **do**
    send estimating request to controllers $C[i]$;
    $\psi_0 \leftarrow$ response time of $C[i]$;
    if $\psi_0 >= 0$ **then**
      $id = i$;
      break;
send the routing request to controller $C[id]$.

Algorithm 2 Queue Management of Controller

Require: the max response delay, RD_{max}.
receive an estimating request from switch s;
calculate the average processing time of requests $\bar{v}$;
if $r_i < \gamma_i$ **then**
  $\psi_i \leftarrow \frac{r_i}{\gamma_i}\bar{v}$;
else
  $\psi_i \leftarrow C[q_i mod \gamma_i] + (\lfloor q_i/\gamma_i \rfloor + 1) \times \bar{v}$;
if $\psi_i > RD_{max}$ **then**
  $\psi_i = -1$;
send ψ_i to switch s.

not exceed limit for the routing request. Last, the switch will send the request to the selected controller.

B. The Selection Models of Controllers

When there are only a few controllers in the network, it is feasible to probe all controllers. Considering that this probing could occupy extra bandwidth of the secure link, it is essential to design a per-flow light-weight probing method.

1) Active Per-Flow Selection of Controllers: To deal with the skew-flow requests and reduce response tail latencies, the switches need to select controllers for each flow. Instead of the controller-to-switch assignment, the active per-flow selection of controllers makes that the routing requests from the same switch can be processed by different controllers. This selection scheme can fully exploit the capabilities of controllers and efficiently reduce response tail latencies.

To reduce the bandwidth consumption of probing controllers, one method is to reduce the number of probed controllers. There is a tradeoff between response tail latencies and the number of probed controllers. Probing more controllers can achieve fewer response tail latencies. However, that also means more bandwidth consumption and computing overhead in controllers. The number of controllers increases as the network scale grows. In this case, checking all controllers has a huge cost. To achieve the light-weight probing, we randomly probe d controllers instead of checking all controllers, where $d \geq 1$. Furthermore, Azar *et al.* [18] have shown that having just two random choices (i.e., $d = 2$) yields a large reduction in the maximum load over having one choice. This method has been widely studied and applied [19]. Inspired by this fact, our active per-flow selection is to probe two controllers and is thus scalable. Meanwhile, the active per-flow selection of controllers can efficiently reduce the bandwidth consumption of the secure link and the computing load of controllers.

Instead of probing the loads of the controllers, probing response delays of controllers can better reduce response tail latencies. The probing of response delays requires that these controllers evaluate their own response delays for routing requests. Since the selection of controllers only needs to get a numerical value of response delay, the selection of controllers is light-weight. Moreover, the heterogeneous and time-varying processing capabilities of controllers increase the complexity of evaluation for response delays of controllers. The response delay estimate model will be introduced in Section IV-C.

2) Active Selection of Controllers for a Batch of Flows: When switches meet bursty-flow requests or when the arrival of routing requests is frequent, conducting a controller selection for each request still aggravates the bandwidth consumption and the loads of controllers even if only two controllers need to be probed in one controller selection. Conducting the controller selection for a batch of arrival routing requests is needed to increase the scalability of the controller selection mechanism, when switches suffer bursty flows.

For active selection of controllers for a batch of flows, the switch conducts one controller selection after it receives the first flow request. When we set the batch size as δ, it means that the following $\delta - 1$ requests will be sent to the same controller with the first request. That is, the result of controller selection for the first request will be shared by δ requests. In addition, the batch selection is irrelevant to the rate of requests because δ denotes the number of requests. Therefore, although there would be the high rate of requests in the beginning when the switch changes its controller, those requests would not be sent to the same controller. Given that the request arrival process at each switch is a Poisson process with rate λ, the arrival duration for a flow is exponentially distributed with mean $1/\lambda$. Therefore, the arrival duration of δ flows is also exponentially distributed with mean δ/λ.

There is a tradeoff between the rounds of controller selection and the performance of controller selection. If δ is too small, it is obvious that controller selection should be frequently conducted. However, it will decrease the performance of controller selection when δ is too large. It is worth noting that it is unnecessary to adopt the batch selection when the arrival of flows is scattered.

C. Queue Management Mechanism of Controller

The queue management of controller makes it so that the selection of controllers can cope and quickly react to heterogeneous and time-varying processing time across controllers. Our queue management mechanism of controller includes response delay estimate and queue length bound.

1) Response Delay Estimate Model: As depicted in Section III-D, the load-aware selection scheme of controllers

IEEE Proof

can not accommodate the heterogeneity of controller. To efficiently reduce the long-tail latency of routing response, switches should select controllers with lower response delays for each routing request.

Request response time consists of the queueing time and the processing time. Furthermore, the queueing time is related to the length of queue, which is equal to the number of queueing requests. Meanwhile, to estimate the queueing time, it is essential to estimate the processing time of each request. In our design, the controller records ν_j, which is the processing time of the latest responded jth requests. Given the number of the latest finished requests s, we calculate $\bar{\nu}$, which denotes the average processing time of s requests in controller i. Thus, $\bar{\nu} = \frac{1}{s}\sum_{j=1}^{s}\nu_j$. We use $\bar{\nu}$ to estimate the processing time of requests.

Consider that the controller can process multiple routing requests simultaneously. We use γ_i to denote the number of requests that $controller_i$ can simultaneously process. q_i and r_i are the number of queueing and running requests in controller i, respectively. To improve the system utilization, the controller that has idle running slots should have a lower estimated response delay. Therefore, the estimated response delay $\psi_i = \frac{r_i}{\gamma_i}\bar{\nu}$ when $r_i < \gamma_i$. When there are requests queueing in a controller, the controller records the running duration $A[k]$ of the running request in the kth slot where $1 \le k \le \gamma_i$. The controller then estimates that the queueing request will run in which slot. To achieve this goal, we use $B[k] = |\bar{\nu} - A[k]|$ and then sort $B[k]$ as non-decreasing order. Then, the controller estimates that the request will run in $[(q_i \mod \gamma_i) + 1]$th slot. We can get the queueing time $wt_i = B[(q_i \bmod \gamma_i)+1]+(\lfloor q_i/\gamma_i \rfloor)\times\bar{\nu}$. At last, $\psi_i = wt_i + \bar{\nu}$ when $r_i = \gamma_i$.

In summary, $controller_i$ uses the following estimation function for response delay:

$$\psi_i = \begin{cases} B[(q_i mod \gamma_i) + 1] + (\lfloor q_i/\gamma_i \rfloor + 1) \times \bar{\nu} & r_i = \gamma_i \\ \frac{r_i}{\gamma_i}\bar{\nu} & r_i < \gamma_i \end{cases} \quad (6)$$

When a switch sends an estimating request to $controller_i$, the $controller_i$ adopts the formula (6) to estimate the response delay. In general, $\gamma_i = 1$ means that the controller only can process one request once. In this case, $\psi_i = B[1]+(q_i+1)\times\bar{\nu}$. We suppose that the controller is empty at the beginning. After that, $\bar{\nu}$ is equal to the average processing time of finished requests when the number of finished messages is lower than the given threshold s.

2) Cutting Tail Latencies: Since switches conduct controller selection simultaneously, there may be "herd behaviors," wherein multiple switches are coaxed to direct requests towards the best controller. There are many requests queueing in a controller under herd behavior that could leads to long-tail latencies of routing responses. Moreover, it is possible that the probed controllers all have low processing capabilities or long queues. In this case, it is not suitable to select a controller from those probed controllers.

To cut long-tail latencies and reduce the influence of herd behavior, the controller necessitates to bound its queue length. Determining the length of queues at controllers is crucial. Queues that are too short lead to lower controller utilization, as resources may remain idle between allocations. Queues that are too long may incur excessive queuing delays.

When fewer requests are sent to a controller, this may incur under-utilization of its resources, whereas significant delays may occur when requests need longer processing time. Hence, after estimating request response delay, we further design a delay-aware bounding mechanism to bound queue length, which can accommodate the heterogeneous and time-varying capabilities of controllers. Meanwhile, bounding queue length can efficiently weaken the influence of herd behaviors. At one point, a controller receives a burst of flows, and that exceeds the limit of queue length. After that, the following flows will not queue in the controller until the queue length is lower than the limit. This delay-aware bounding mechanism relies on the response delay estimation of request, which is reported by the controller.

In particular, we specify the maximum response delay RD_{max} that a request is allowed to wait in a queue. When we are about to place a request at the queue of $controller_i$, we first check the estimated response delay ψ_i reported by $controller_i$. Only if $\psi_i < RD_{max}$ is the request queued at that controller. We sample d controllers while conducting the controller selection. If the d selected controllers all do not satisfy the maximum response delay constraint. The switch needs to reselect a new controller. Using this method, the number of requests in each controller gets dynamically adapted based on the current capability of the controller.

RD_{max} is set to make requests prefer faster controllers. Furthermore, RD_{max} can be dynamically regulated to fit the variation of controllers' capabilities. For controllers that have low processing capabilities, RD_{max} can limit the number of queueing requests in these controllers. After that, these requests can be sent to faster controllers. However, if RD_{max} is too small, most of controllers refuse to receive new requests because their response delays exceed the limit of RD_{max}. In this case, it is necessary to extend the value of RD_{max}.

V. Performance Evaluation

We start with the evaluation methodology and scenarios. In this section, we evaluate the selection schemes of controller and the assignment mechanism of controller under the general settings of controllers, the queue length bound RD_{max}, the heavy request-skews, the time-varying service rates and the batch selection.

A. Experimental Setup

We build a discrete-event simulator wherein workload generators create flow requests at a set of switches. These switches then employ the controller selection scheme to select a controller for each request. Unless otherwise specified, the network consists of 300 switches and 40 controllers. However, when a large amount of flow requests emerges, we need to employ more controllers to deal with the burstly flows. The workload generators create flow requests according to a Poisson arrival process to mimic arrival of user requests at web servers [20]. Unless otherwise specified, the Poisson

IEEE Proof

arrival process is with $\lambda = 0.5$ during 1ms. At the beginning, the system is empty, and there are no requests. With the system running, the switches start to produce flow requests and select controllers for flow requests. We run the system 1000ms, and the number of arrival flow requests is about 150, 000. Each controller maintains a FIFO request queue. Moreover, in the settings of controllers, each controller can service a tunable number of requests in parallel (10 in our settings). The processing time each request experiences is drawn from an exponential distribution (as in [21]) with a mean processing time $\mu^{-1} = 2$ms. Furthermore, we incorporate controller heterogeneity into the network as follows: each controller, independently and with a uniform probability, sets its service rate to a different μ, where $\mu = 0.5$ or $\mu = 0.1$ in our settings. To estimate the response delay of each request, we set $s = 100$, That is, $\bar{\nu}$ denotes the average processing time of the latest finished 100 requests. We repeat every experiment 20 times using different random seeds, and then get the average result.

We compare our design against three strategies:

1) **Quantity-based allocation**: Controllers achieve load balance through balancing the number of switches, which each controller manages. Currently, ONOS controller utilizes this strategy to balance the loads of distributed controllers.
2) **Load-aware selection**: The switch probes two controllers for each request and sends the request to the controller with fewer number of requests because probing all controllers is not scalable.
3) **Delay-aware selection**: The switch probes two controllers for each request and gets request response delays, which rely on the feedbacks of probed controllers. Then, the switch sends the request to the controller with smaller response delay.

B. The Impact of d

Azar *et al.* [18] have shown that the situation of $d = 2$ yields a large reduction in the maximum load over $d = 1$, while each additional choice beyond two decreases the maximum load by just a constant factor. Further, to verify the theory and determine the value of d, we evaluate the impact of d on the performance of the delay-aware selection strategy under heterogeneous controllers where d denotes the number of probed controllers. The processing time of each request in each controller is drawn from an exponential distribution where μ was randomly set as 0.5 or 0.1. Other parameters are the same as Section V-A.

Fig. 6 shows the delay-aware selection strategy significantly reduces the mean response time as the value of d increase from 1 to 2. However, probing more controllers just incur a little bit reduction on the mean response time after d exceeds 2. In addition, multiple switches may simultaneously select the same controller if each of them randomly probes more controllers. This would in turn aggravates the load of the controller. Thus, we set $d = 2$ in the next experiments.

C. General Settings of Controllers

We evaluate the performances of different schemes with heterogeneous controllers. We employ 60 controllers where the

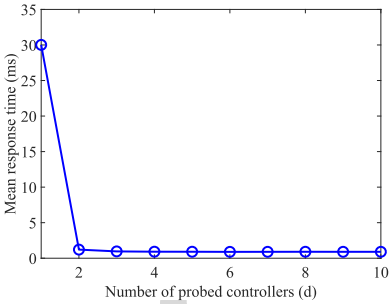
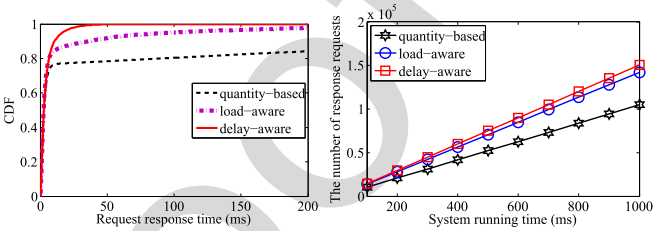


Fig. 6. The impact of d on the performance of the delay-aware selection strategy.



(a) Request response time with different (b) Throughput under different schemes.

Fig. 7. The performances of different schemes where controllers are heterogeneous.

bound of maximal response delay $RD_{max} = 20$ms. The other settings of experiments are consistent with that of Section V-B.

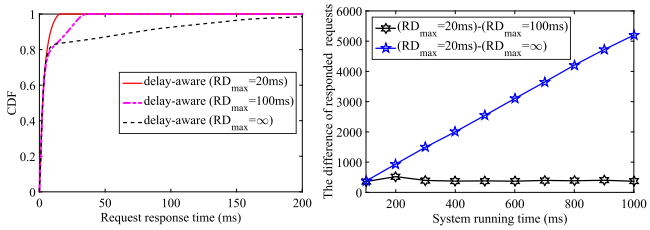
Fig. 7(a) shows that our delay-aware scheme can significantly reduce the response tail latencies. Basically, all requests can be finished in 50ms while adopting our delay-aware scheme. The load-aware scheme achieves better performance than the quantity-based scheme in Fig. 7(a). Over 90% of requests can be processed during 150ms based on the load-aware scheme. The quantity-based scheme leads to long response delays, and there are over 20% of requests whose response delays are more than 200ms in Fig. 7(a). This is because a large of requests queue in controllers that have lower processing capabilities. Meanwhile, the load-aware scheme also failed to respond to requests quickly. The processing delays of flow requests in different controllers are different when controllers are heterogeneous. As a consequence, selecting a controller by the number of requests is not efficient. Fig. 7(a) reveals that our delay-aware scheme achieved the lowest response duration due to not only estimating response delay but also cutting tail latencies. Fig. 7(a) also reveals that our delay-aware scheme can be adapted to the system where controllers have heterogeneous capabilities.

Fig. 7(b) shows that our delay-aware scheme can respond to more requests than the load-aware scheme and the quantity-based scheme can in the same time period. Meanwhile, the throughput difference among schemes grows as the system runs. In summary, with heterogeneous controllers, our delay-aware scheme can efficiently reduce response tail latencies and improve the throughput of controllers.

D. Impact of Queue Length Bound RD_{max}

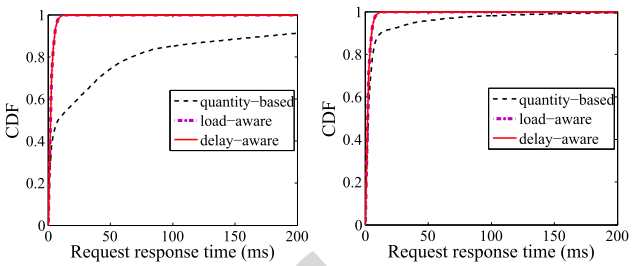
We evaluate the impact of RD_{max} on the performance of the delay-aware scheme. We set $RD_{max} = 20$ms, $RD_{max} = 100$ms and $RD_{max} = \infty$ respectively. $RD_{max} = \infty$ means

IEEE Proof



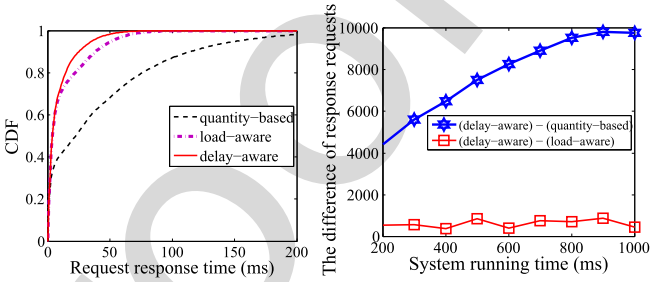
(a) The performance of delay-aware scheme under different RD_{max} settings.(b) The difference of responded requests under different RD_{max} settings.

Fig. 8. The impact of RD_{max} on the performance and throughput of the delay-aware scheme.



(a) $Request\text{-}skew$=20%. (b) $Request\text{-}skew$=50%.

Fig. 9. Request response time with different schemes under the heavy request-skews.



(a) Request response time with different schemes. (b) The difference of responded requests between different schemes.

Fig. 10. The performances of different schemes under the time-varying service rates.

that there is no limit on the queue length of controllers. Other parameters are the same with Section V-C.

Fig. 8(a) reveals that our delay-aware scheme has better performance when RD_{max} has a smaller value. Under $RD_{max} = 20$ms, all requests can be finished in 20ms. However, The maximal response delay is 40ms under $RD_{max} = 100$ms. Therefore, the performance of delay-aware scheme under $RD_{max} = 20$ms is better than that of $RD_{max} = 100$ms. Comparing with $RD_{max} = \infty$, delay-aware scheme can significantly reduce response tail latencies when $RD_{max} = 20$ms. Furthermore, we compare the throughput of controllers under different RD_{max} settings. Fig. 8(b) shows that the system can respond to 300 more requests under $RD_{max} = 20$ms than $RD_{max} = 100$ms. It was obvious that controllers have higher throughput when $RD_{max} = 20$ms than when $RD_{max} = 100$ms and $RD_{max} = \infty$. The difference of responded requests between $RD_{max} = 20$ms and $RD_{max} = 100$ms remains stable. However, the difference of responded requests between $RD_{max} = 20$ms and $RD_{max} = \infty$ grows as the system runs.

Fig. 8 reveals that the setting of RD_{max} can make flow requests prefer the controllers that have faster processing capabilities. Additionally, it is noteworthy that RD_{max} can not be set too small, otherwise, there are no available controllers while conducting the controller selection.

E. Performance Under Heavy Request-Skews

In this section, we study the effect of heavy demand skews on the observed latencies where controllers are homogeneous with average server rate $\mu = 0.5$. We set request-skew= 20% and request-skew= 50%. That is, 20% and 50% of switches generated 80% of the total requests towards the controllers. Most of parameters are inherited from Section V-A. To enable 20% of switches to generate 80% of the total requests, we randomly select 60 switches and set the arrival rate of flow requests $\lambda = 2$. Other switches set $\lambda = 0.125$. Under request-skew= 50%, half of the switches set $\lambda = 0.8$, and the other half of the switches set $\lambda = 0.2$. We set $RD_{max} = 150$ms because there are too many requests in a short time and these requests have to queue in controllers.

Fig. 9(a) shows that over 5% requests have response delays of more than 200ms for the quantity-based scheme. The quantity-based scheme suffers decreased performance due to the request-skews. However, the load-aware and delay-aware schemes can significantly reduce response tail latencies. Based on the load-aware and delay-aware schemes, all requests can be finished in 10ms in Fig. 9(a). Fig. 9 reveals that the load-aware and delay-aware schemes achieve very similar performances since controllers are homogeneous in this section. Meanwhile, the quantity-based scheme suffers decreased performance due to the request-skews. Under request-skews, a part of switches generate a large number of the requests, which incur long queues in some controllers for the quantity-based scheme.

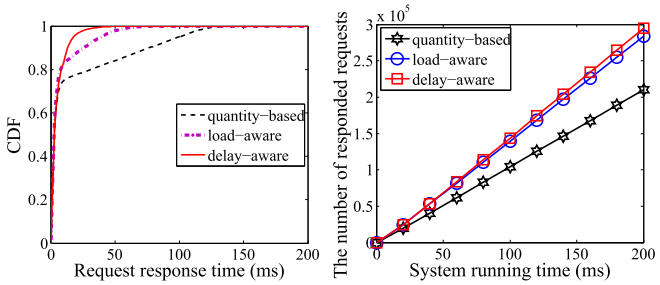
Comparing Fig. 9(a) and Fig. 9(b), we can find that the quantity-based scheme under request-skew= 50% achieves a lower response latency than under request-skew= 20%. It is because the load balance among controllers where request-skew= 50% is better than that of request-skew= 20%. Moreover, Fig. 9 also reveals that our delay-aware scheme can accommodate the heavy request-skews.

F. Impact of Time-Varying Service Rates

In this section, we study the effect of the service rate fluctuation on the tail latency of response. We change the average service rates of controllers in the system every 50ms, and all controllers randomly set $\mu = 0.5$ or $\mu = 0.1$. Other parameters are inherited from Section V-C.

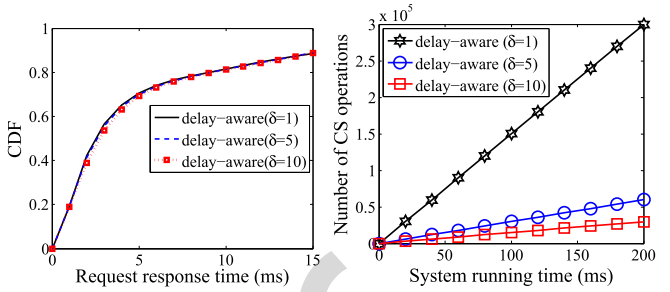
Fig. 10(a) reveals that our delay-aware scheme can respond to all requests in 60ms, the load-aware scheme can respond to all requests during 80ms, and the response tail latency of the quantity-based scheme is more than 200ms. Therefore, our delay-aware scheme can efficiently reduce response tail latencies. For the quantity-based scheme, it could not exploit the feedbacks of controllers to select the controller and further

IEEE Proof



(a) Request response time with different schemes. (b) Throughput under different schemes.

Fig. 11. The performances of different schemes under the batch selection.



(a) The performance of delay-aware scheme. (b) The number of CS operations. CS: the selection of controller.

Fig. 12. The impact of the batch size δ.

suffers lower system utilization. The load-based scheme also suffers lower performance because it fails to consider the time-varying service rate. Fig. 10(a) reveals that our delay-based scheme can accommodate time-varying service rate.

Fig. 10(b) shows that our delay-aware scheme can respond to more requests than the quantity-based scheme and the load-aware scheme. With the increase of system running time, the advantage of the delay-aware scheme is more obvious than the quantity-based scheme in Fig. 10(b). Meanwhile, the difference of responded requests between the delay-aware scheme and the load-aware scheme cyclically fluctuates because the service rates of controllers are periodically changed. In summary, our delay-aware scheme can be better adapted to the time-varying service rate and can efficiently reduce tail latencies of responses.

G. Evaluation for Batch Selection

In this section, we evaluate the performance of different schemes for batch arrival requests. We set the arrival rate of flow requests $\lambda = 5$. There are about $300,000$ requests during 200ms. To deal with the burst requests, we employ 150 controllers, and each controller can process 40 requests in parallel. We set the batch size $\delta = 10$ and $RD_{max} = 15$ms, and other parameters are set as Section V-C.

Fig. 11(a) reveals that our delay-aware scheme can still reduce tail latencies of responses under the batch selection. Our delay-aware scheme can respond to all requests in 40ms, and the load-aware scheme can respond to all requests during 60ms. However, the response tail latency of the quantity-based scheme is more than 130ms. The performance of the quantity-based scheme is mainly affected by controllers that have lower processing capabilities. The tail latency of response under the load-aware scheme is generated because it fails to accommodate the heterogeneous controllers. Meanwhile, Fig. 11(b) shows that our delay-aware scheme can respond to more flow requests than either of the other two schemes. This means that our delay-aware scheme can not only reduce the tail latency, but can also improve the throughput of controllers under the batch selection.

Impact of Batch Size δ: Furthermore, we evaluate the impact of the batch size δ on the performance of the delay-aware scheme. Fig. 12(a) depicts the performances of the delay-aware scheme under different batch sizes. Fig. 12(a) shows that the performance of the delay-aware scheme has a modest decrease with the increase of the batch size δ. When the batch size δ increases, it means that more requests will be sent to the same controller, even though the controller has a low processing capacity. As a consequence, the delay-aware scheme suffers a little of decreased performance. we can find that experiments show a similar performance under the different settings of δ in Fig. 12(a). Although more requests would be directed to the controller with a low processing capacity at some time when the value of δ is larger, after that, the following requests would have less chance to select the controller. Therefore, the decrease of performance is modest. Furthermore, Fig. 12(b) reveals that the number of controller selection operations dramatically decreases when the batch size δ goes up. The fewer controller selection operations in switch end will incur less bandwidth consumption in the secure links between switches and controllers and less computing load in controllers. In conclusion, active selection of controllers for a batch of flows can efficiently reduce the resourse consumption of communication and computing with a little bit of performance reduction.

VI. Related Work

A. Network Softwarization

Network softwarization is a transformation trend for designing, implementing, and managing the 5G and next generation networks. It exploits the benefits of software to enable the redesign of network and service architectures, optimize the expenditure and operational costs, and bring new values in the infrastructures. The key enablers consist of the network function virtualization (NFV), software-defined networking (SDN) and cloud computing, etc. Meanwhile, 5G systems will also rely on these technologies to attain the systems flexibility and elasticity [2].

Along with recent and ongoing advances in cloud computing, it has become promising to design flexible, scalable, and elastic 5G systems benefiting from advanced virtualization techniques of cloud computing [1]. Taleb and Ksentini [22] introduces the Follow-Me Cloud concept and proposes its framework. There has been research on the possibility of extending cloud computing beyond data centers toward the mobile end user, providing end-to-end mobile connectivity as a cloud service [23]. Software defined networking (SDN)

IEEE Proof

acts as a promising enabler for network softwarization and plays a crucial role in the design of 5G wireless networks [1]. SDN has been also utilized in the virtualization of mobile network functions [24]. Kempf *et al.* [25] describe an evolution of the mobile Evolved Packet Core (EPC) utilizing SDN that allows the EPC control plane to be moved into a data center. Taleb *et al.* [1] introduce the concept of "Anything as a Service" (ANYaaS), which relies on the reference ETSI NFV architecture to orchestrate and manage important services. NFV aims at offering network services using network functions implemented in software and deployed in an on-demand and elastic manner on the cloud [26]. Medhat *et al.* [5] introduce a service function chaining taxonomy as the basis for the subsequent state-of-the-art analysis.

B. SDN Scalability

To tackle the problem of scaling SDN, Xie *et al.* [27] resort to design the distributed controllers, such as Onix [28] and ONOS [14], which try to distribute the control plane while maintaining a logically centralized management. These approaches balance the load of controllers based on the number of switches, which can not efficiently reduce the tail latencies of responses. Aissioui *et al.* [29] propose a two-level hierarchical controller platform to address these scalability and performance issues in the context of 5G mobile networks.

One key limitation of the distributed controllers is that the mapping between a switch and a controller is statically configured. The static configuration results in the uneven load distribution among the controllers. Bari *et al.* [30] propose a management framework, which periodically evaluates the current controller-to-switch assignment. After that, it needs to decide whether to perform a reassignment. If a reassignment is performed, the management framework also changes the controller-to-switch assignment in the network. Dixit *et al.* [31] propose ElastiCon, an elastic distributed controller architecture in which the controller pool is dynamically grown or shrunk according to traffic conditions. In this case, the load is dynamically shifted across controllers, which similarly relieves the static mapping between a switch and a controller. Zhou *et al.* [32] propose a dynamic and adaptive algorithm (DALB), which is running as a module of SDN controller. The controllers in distributed environment can cooperate with each other to keep load balancing. Overloaded controller can be detected, and high-load switches mapped to this controller can be smoothly migrated to the under-load controllers. However, these dynamic frameworks require that the control plane monitors the state of the whole network and conducts the reassignments, which aggravate the computing load of the control plane.

In contrast to these works, our approach relies on simple and inexpensive feedback of controllers and efficiently relieve the load of the control plane. Mao and Shen [33] use the principles of SDN to achieve the server load balancing by setting the SDN flow table, which does not aim to solve the load balance of the distributed controllers of SDN. Palma *et al.* [34] develop the QueuePusher, which is a queue management extension to OpenFlow controllers supporting the Open vSwitch Database Management Protocol (OVSDB) standard. QueuePusher can generate the appropriate queue configuration messages for switches. In addition, there have been lots of researches on how to achieve the consistency among distributed controllers [15]. These techniques are complementary to our approach.

VII. Conclusion

NFV and SDN can dynamically redistribute the flow across appropriate VNFs or service function chains if the controller configures a desired routing path for each network flow resulting from NFV applications. In this paper, we present the long-tail observations of the routing response latencies while using the up-to-date controller-to-switch assignment mechanism. To tackle this essential problem, we first propose a light-weight and load-aware switch-to-controller selection scheme to cut the long-tail response latency under the simple scenario of homogeneous controllers, and then design a general delay-aware switch-to-controller selection scheme to fundamentally cut the long-tail response latency for the more complicated heterogeneous controller scenario with performance fluctuations. Through comprehensive performance evaluation, we demonstrate that our adaptive controller selection schemes can efficiently reduce response long-tail latencies and accommodate various system environments including the request-skews, the fluctuation of service rates and so on.

References

[1] T. Taleb, A. Ksentini, and R. Jantti, "'Anything as a service' for 5G mobile systems," *IEEE Netw.*, vol. 30, no. 6, pp. 84–91, Nov. 2016.

[2] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, "Mobile edge computing potential in making cities smarter," *IEEE Commun. Mag.*, vol. 55, no. 3, pp. 38–43, Mar. 2017.

[3] Y. Zhou, D. Zhang, and N. Xiong, "Post-cloud computing paradigms: A survey and comparison," *Tsinghua Sci. Technol.*, vol. 22, no. 6, pp. 714–732, 2017.

[4] M. Kablan, A. Alsudais, E. Keller, and F. Le, "Stateless network functions: Breaking the tight coupling of state and processing," in *Proc. 14th USENIX NSDI*, Boston, MA, USA, Mar. 2017, pp. 97–112.

[5] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz, "Service function chaining in next generation networks: State of the art and research challenges," *IEEE Commun. Mag.*, vol. 55, no. 2, pp. 216–223, Feb. 2017.

[6] P. Zave, R. A. Ferreira, X. K. Zou, M. Morimoto, and J. Rexford, "Dynamic service chaining with dysco," in *Proc. ACM SIGCOMM*, Los Angeles, CA, USA, 2017, pp. 57–70.

[7] D. L. C. Dutra, M. Bagaa, T. Taleb, and K. Samdanis, "Ensuring end-to-end QoS based on multi-paths routing using SDN technology," in *Proc. IEEE Global Commun. Conf.*, Dec. 2017, pp. 1–6.

[8] N. Gude *et al.*, "NOX: Towards an operating system for networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008.

[9] A. Ksentini, M. Bagaa, T. Taleb, and I. Balasingham, "On using bargaining game for optimal placement of SDN controllers," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–6.

[10] J. Xie, D. Guo, Z. Hu, T. Qu, and P. Lv, "Control plane of software defined networks: A survey," *Comput. Commun.*, vol. 67, pp. 1–10, Aug. 2015.

[11] D. Kreutz, F. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[12] A. Ksentini, M. Bagaa, and T. Taleb, "On using SDN in 5G: The controller placement problem," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–6.

[13] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, 2008.

IEEE Proof

[14] P. Berde *et al.*, "ONOS: Towards an open, distributed SDN OS," in *Proc. ACM HotSDN*, 2014, pp. 1–6.
[15] A. Panda, W. Zheng, X. Hu, A. Krishnamurthy, and S. Shenker, "SCL: Simplifying distributed SDN control planes," in *Proc. 14th USENIX NSDI*, Mar. 2017, pp. 329–345.
[16] D. Hugheshallett, A. M. Gleason, and W. G. Mccallum, *Calculus: Single and Multivariable, Student Solutions Manual*, 6th ed. Hoboken, NJ, USA: Wiley, 2013.
[17] A. Bremler-Barr, Y. Harchol, and D. Hay, "OpenBox: A software-defined framework for developing, deploying, and managing network functions," in *Proc. ACM SIGCOMM*, Salvador, Brazil, Aug. 2016, pp. 511–524.
[18] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal, "Balanced allocations," *SIAM J. Comput.*, vol. 29, no. 1, pp. 180–200, 1999.
[19] M. Mitzenmacher, A. W. Richa, and R. Sitaraman, "The power of two random choices: A survey of techniques and results," in *Handbook of Randomized Computing*, vol. 11. Norwell, MA, USA: Kluwer, 2000, pp. 255–312.
[20] R. Nishtala *et al.*, "Scaling memcache at Facebook," in *Proc. USENIX NSDI*, 2013, pp. 385–398.
[21] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," in *Proc. ACM Conf. Emerg. Netw. Experim. Technol.*, 2013, pp. 283–294.
[22] T. Taleb and A. Ksentini, "Follow me cloud: Interworking federated clouds and distributed mobile networks," *IEEE Netw.*, vol. 27, no. 5, pp. 12–19, Sep./Oct. 2013.
[23] T. Taleb, "Toward carrier cloud: Potential, challenges, and solutions," *IEEE Wireless Commun.*, vol. 21, no. 3, pp. 80–91, Jun. 2014.
[24] T. Taleb, A. Ksentini, and A. Kobbane, "Lightweight mobile core networks for machine type communications," *IEEE Access*, vol. 2, pp. 1128–1137, 2014.
[25] J. Kempf, B. Johansson, S. Pettersson, H. Lüning, and T. Nilsson, "Moving the mobile evolved packet core to the cloud," in *Proc. IEEE 8th Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, Oct. 2012, pp. 784–791.
[26] T. Taleb *et al.*, "EASE: EPC as a service to ease mobile core network deployment over cloud," *IEEE Netw.*, vol. 29, no. 2, pp. 78–88, Mar./Apr. 2015.
[27] J. Xie, D. Guo, X. Zhu, B. Ren, and H. Chen, "Minimal fault-tolerant coverage of controllers in IaaS datacenters," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2017.2753260.
[28] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *Proc. USENIX OSDI*, 2010, pp. 351–364.
[29] A. Aissioui, A. Ksentini, A. M. Gueroui, and T. Taleb, "Toward elastic distributed SDN/NFV controller for 5G mobile cloud management systems," *IEEE Access*, vol. 3, pp. 2055–2064, 2015.
[30] M. F. Bari *et al.*, "Dynamic controller provisioning in software defined networks," in *Proc. Int. Conf. Netw. Service Manage.*, Oct. 2013, pp. 18–25.
[31] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," in *Proc. ACM HotSDN*, Hong Kong, Aug. 2013, pp. 7–12.
[32] Y. Zhou *et al.*, "A load balancing strategy of sdn controller based on distributed decision," in *Proc. IEEE 13th Int. Conf. Trust, Secur. Privacy Comput. Commun.*, Sep. 2014, pp. 851–856.
[33] Q. Mao and W. Shen, "A load balancing method based on SDN," in *Proc. 7th Int. Conf. Meas. Technol. Mechatronics Autom.*, Jun. 2015, pp. 18–21.
[34] D. Palma *et al.*, "The queuepusher: Enabling queue management in openflow," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, Sep. 2014, pp. 125–126.

Junjie Xie received the B.S. degree in computer science and technology from the Beijing Institute of Technology, Beijing, China, in 2013, and the M.S. degree in management science and engineering from the National University of Defense Technology, Changsha, China, in 2015, where he is currently pursuing the Ph.D. degree. He is also a joint Ph.D. student with the University of California, Santa Cruz (UCSC), CA, USA, from 2017. His study in the UCSC is supported by the China Scholarship Council. His research interests include distributed systems, data-center networks, and software-defined networks.

Deke Guo received the B.S. degree in industry engineering from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 2001, and the Ph.D. degree in management science and engineering from the National University of Defense Technology, Changsha, China, in 2008. He is currently a Professor with the College of Information System and Management, National University of Defense Technology, and a Professor with the School of Computer Science and Technology, Tianjin University. His research interests include distributed systems, software-defined networking, data center networking, wireless and mobile systems, and interconnection networks. He is a member of the ACM.

Xiaozhou Li received the Ph.D. degree in computer science from Princeton University in 2016. He is currently a Software Engineer with Barefoot Networks, where he is building new networking systems with fast programmable network chips. His research improves the performance, scalability, and efficiency of datacenter services, with a particular focus on combining new hardware and infrastructure capabilities with careful architectural design and algorithm engineering.

Yulong Shen received the B.S. and M.S. degrees in computer science and the Ph.D. degree in cryptography from Xidian University, Xian, China, in 2002, 2005, and 2008, respectively. He is currently a Professor with the School of Computer Science and Technology, Xidian University, and also an Associate Director of the Shaanxi Key Laboratory of Network and System Security. He has also served on the technical program committees of several international conferences, including the NANA, the ICEBE, the INCoS, the CIS, and the SOWN. His research interests include wireless network security and cloud computing security.

Xiaohong Jiang received the B.S., M.S., and Ph.D. degrees from Xidian University, China, in 1989, 1992, and 1999 respectively. From 2005 to 2010, he was an Associate professor, Tohoku University. He is currently a Full Professor with Future University Hakodate, Japan. His research interests include computer communications networks, mainly wireless networks and optical networks, network security, and routers/switches design. He has authored or co-authored over 300 technical papers at premium international journals and conferences, which include over 70 papers published in top IEEE journals and top IEEE conferences. He was a recipient of the Best Paper Award of IEEE HPCC 2014, IEEE WCNC 2012, IEEE WCNC 2008, IEEE ICC 2005-Optical Networking Symposium, and IEEE/IEICE HPSR 2002. He is a Member of IEICE.

IEEE PROOF

Cutting Long-Tail Latency of Routing Response in Software Defined Networks

Junjie Xie, Deke Guo, Xiaozhou Li, Yulong Shen, and Xiaohong Jiang

***Abstract*— To enable the network softwarization, network function virtualization (NFV) and software defined networking (SDN) are integrated to jointly manage and utilize the network resource and virtualized network functions (VNFs). For a network flow resulting from any NFV application, an associated switch would send a routing request to the controller in SDN. The controller then generates and configures a routing path to dynamically steer the flow across appropriate VNFs or service function chains. This process, however, exhibits a skew distribution of response latency with a long tail. Cutting the long-tail latency of response is critical to enable the network softwarization, yet difficult to achieve due to many factors, such as the limited capacities and the load imbalance among controllers. In this paper, we reveal that such flow requests still experience the long-tail response latency, even using the up-to-date controller-to-switch assignment mechanism. To tackle this essential problem, we first propose a light-weight and load-aware switch-to-controller selection scheme to cut the long-tail response latency under the simple scenario of homogeneous controllers, and then design a general delay-aware switch-to-controller selection scheme to fundamentally cut the long-tail response latency for the more complicated heterogeneous controller scenario with performance fluctuations. The comprehensive evaluations indicate that our two new switch-to-controller selection schemes can significantly reduce the long-tail latency and provide higher system throughput.**

***Index Terms*— Network softwarization, software defined networks, controller selection, long-tail latency.**

Manuscript received September 30, 2017; revised February 5, 2018; accepted February 27, 2018. This work was supported in part by the National Natural Science Foundation for Outstanding Excellent Young Scholars of China under Grant 61422214, in part by the National Natural Science Foundation of China under Grant 61772544 and Grant U1536202, in part by the National Basic Research Program (973 program) under Grant 2014CB347800, in part by the Hunan Provincial Natural Science Fund for Distinguished Young Scholars under Grant 2016JJ1002, and in part by the Guangxi Cooperative Innovation Center of cloud computing and Big Data under Grant YD16507 and Grant YD17X11. *(Corresponding authors: Deke Guo; Yulong Shen.)*

J. Xie is with the Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha 410073, China (e-mail: xiejunjie06@gmail.com).

D. Guo is with the College of System Engineering, National University of Defense Technology, Changsha 410073, China. He is also with the School of Computer Science and Technology, Tianjin University, Tianjin 300072, China (e-mail: guodeke@gmail.com).

X. Li is with the Department of Computer Science, Princeton University, Princeton, NJ 08544 USA (e-mail: xl@cs.princeton.edu).

Y. Shen is with the School of Computer Science and Technology, Xidian University, Xi'an 710071, China (e-mail: ylshen@mail.xidian.edu.cn).

X. Jiang is with the School of Systems Information Science, Future University Hakodate, Hokkaido 041-8655, Japan (e-mail: jiang@fun.ac.jp).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2018.2815358

I. Introduction

NETWORK softwarization is a transformation trend for designing, implementing, and managing the next generation networks. It exploits the benefits of software to enable the redesign of network and service architectures, optimize the expenditure and operational costs, and bring added values. The key enablers consist of the network function virtualization (NFV), software-defined networking (SDN) and cloud computing, etc [1]. Moreover, 5G systems will also rely on these technologies to attain system's flexibility and true elasticity [2], [3]. Network functions (NFs) are crucial for improving network security by examining and modifying network flows using special-purpose hardware. Recently, NFV has been proposed to execute virtual network functions (VNFs) on generic compute resources [4], such as commodity servers and VMs. Normally, a flow goes through specific VNFs in a particular order to meet its required processing, following the service function chain (SFC) [5], [6] along a routing path.

Additionally, SDN offers the freedom to refactor the control plane and flexibly enables the network softwarization [7]. More precisely, NFV and SDN can jointly manage the network resource and VNFs, and dynamically steer network flows across appropriate VNFs or SFCs. SDN centralizes the network control plane to a programmable software component, i.e., a controller running on a generic server, such as NOX [8]. The controller maintains a global network view and optimizes the forwarding decisions of network flows. For a flow from any NFV application, an associated switch would send a routing request to the controller. It is the controller that generates and configures a routing path to a specific VNF instance or to traverse a SFC on demand. The above process between a pair of switch and controller brings the response latency. Many factors would skew the tail of the latency distribution. For example, a single controller that lacks sufficient capacity to tackle received routing requests quickly and inevitably becomes a performance bottleneck [9]. Thus, such routing requests experience long-tail latency of response, as evaluated in Section II. Cutting the long-tail latency of routing response is critical to enable the network softwarization, yet difficult to achieve due to many factors.

To improve the scalability of SDN, the distributed control plane consisting of multiple controllers has been proposed recently [10], such as ONOS and OpenDaylight. To reduce the long-tail latency of response, they resort to the controller-to-switch assignment mechanism. That is, the control plane

IEEE Proof

0733-8716 © 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

proactively assigns a controller to each switch such that each controller manages the same amount of switches. In reality, the quantities of routing requests coming from switches per unit time are different and dynamic. Consequently, controllers still differ in the amount of received routing requests per time unit. This load imbalance among controllers leads to the long-tail latency of response. Additionally, the controller-to-switch assignment requires coordination among controllers, which further aggravates the loads of controllers.

In this paper, to cut the long-tail latency of response and lighten the load of controllers, we propose conducting the selection of controllers at the side of each switch instead of the controller-to-switch assignment. This means that each switch actively chooses one controller from multiple available ones, decoupling the static binding between switches and controllers. More precisely, each switch prefers to adaptively select the controller with low response latency for routing requests. This would move the partial intelligences of the network to switches and efficiently reduces the loads of controllers.

Despite those potential benefits, the selection of controllers still faces many challenges. First, the switches need to probe the state of controllers via the secure channel between them. The secure channel is one kind of rare resource and affects the performance of the whole network. To save the bandwidth of the secure channel, the selection process of controllers should be light-weight and use a few of the feedbacks from the controllers. Second, the selection scheme needs to be scalable, irrespective of the network size and the number of controllers. Third, the selection scheme needs to accommodate the bursty and skew routing requests from switches. Last, the selection scheme should adapt to the heterogeneous controllers and the performance fluctuation across controllers.

To tackle such challenges, we design a *load-aware* selection scheme of controllers, which is simple but effective to achieve the load balance among controllers. The load of a controller refers to the number of routing requests waited to be processed. The basic idea of our scheme is that each switch sends routing requests to the controller with the lowest load. In this way, all controllers process the similar number of routing requests per time slot. This is very helpful to cut the long-tail latency of routing response, when all controllers have the same processing capabilities. This method alone, however, is insufficient to deal with more general settings of heterogenous controllers and the performance fluctuation. Those controllers with lower processing capabilities still incur the long-tail latency of response for routing requests, when all controllers achieve the load balance. For this reason, we further present a general *delay-aware* selection scheme of controllers to fundamentally cut the long-tail latency of routing response.

Our *delay-aware* selection scheme includes two key components. The first one is the controller selection model of each switch, which uses simple and inexpensive probing feedbacks from a few controllers. It is still effective if each switch just randomly probes two controllers and sends upcoming routing requests to the controller with the shorter response delay. This model is scalable and light-weight since it is not affected by the network scale and the number of controllers. The second component is the queue management mechanism of each controller. It could estimate the response delay of a routing request and hence improve the performance predictability of controllers. The evaluation results reveal that our *delay-aware* selection scheme can efficiently reduce the long-tail latency of routing responses and improve the system throughput.

In summary, the major contributions of this paper are as follows.

1) We reveal that routing requests experience the long-tail response latency, even using the up-to-date controller-to-switch assignment mechanism in SDN. Therefore, we propose an adaptive selection mechanism of controllers for switches to cut the long-tail latency.
2) We first design an efficient *load-aware* selection method of homogeneous controllers for each switch. For more general scenarios, we further propose a general *delay-aware* selection method, which is adaptive to the bursty routing requests, the heterogenous controllers and the performance fluctuations. Our two methods are light-weight as they limit the additional overhead caused by probing two controllers.
3) We further develop a queue management mechanism for each controller, which can efficiently manage the queue length and estimate the response delay of routing requests. The evaluation results reveal that our controller selection methods can accommodate system environment variations and efficiently reduce long-tail latency of routing response.

The paper is organized as follows. In Section II, we present the observation of long-tail latency of routing response. Section III depicts the framework of our controller selection mechanism and the load-aware selection scheme of controllers. We present the delay-aware selection scheme of controllers in Section IV. We conduct massive experiments to evaluate the performance of our controller selection schemes under various system environment in Section V. Section VI introduces the related work. In Section VII, we conclude this paper.

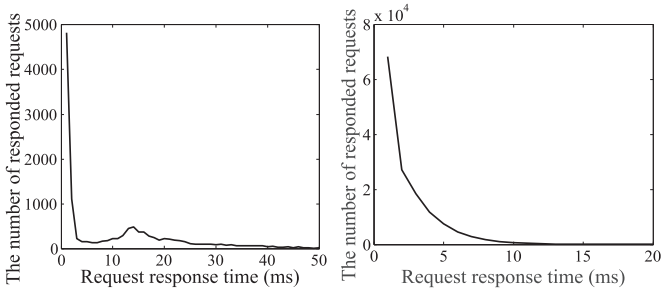
II. Long Tail of Response Latencies

In a SDN, when a switch receives a new flow, the switch sends a routing request to its controller. The controller then computes a route for the flow and inserts flow rules to related switches in the route. Thus, the new flow would be forwarded according to the flow rules in switches [11], [12]. Such an interaction between the switch and the controller causes the response latency. For a routing request, the latency of routing response denotes the time interval from sending the routing request to receiving the flow rules generated by the controller.

A. Long-Tail Observations of Response Latencies

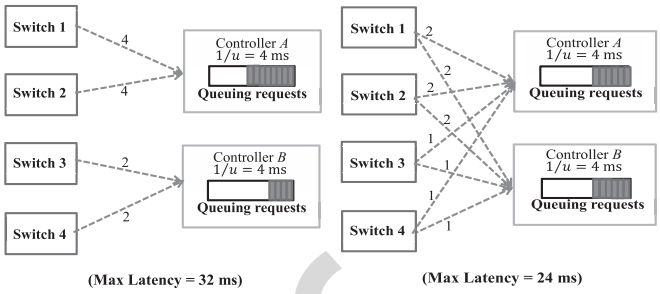
Fig. 1(a) plots an observation about the long-tail distribution under a single instance of ONOS controller. We build a SDN testbed with one controller, running in a virtual machine with 2 CPU cores and 2G RAM. Note that the testbed forms a typical Fat-tree datacenter topology [13]. We record the response latencies of 12,000 routing requests. As shown in Fig. 1(a), the response latencies of 50% of routing requests are lower than 5ms, and 90% of routing requests are

IEEE Proof



(a) A single controller with 12,000 routing requests. (b) Multiple controllers with 150,000 routing requests.

Fig. 1. Long-tail distributions of routing responses under a single controller as well as multiple controllers.



(a) Quantity-based controller allocation. (b) Load-aware controller selection.

Fig. 2. The performances of quantity-based controller allocation and load-aware controller selection.

served within 30ms. However, there still exist some routing requests whose response latencies are more than 50ms. That is, the response latencies exhibit a long-tail distribution.

Furthermore, we observe the response latencies of routing requests under multiple controllers [14]. We employ 300 switches and 40 controllers where each controller manages 7 or 8 switches. Each switch generates routing requests according to a Poisson arrival process with $\lambda = 0.5$ during 1ms. The processing time of each request in each controller is drawn from an exponential distribution where $\mu^{-1} = 2$ms. Each controller can process 10 requests in parallel. We run the system 1000ms and record the response latencies of routing requests. The number of arrival routing requests is about $150,000$. In Fig. 1(b), 89% of routing requests can be served in 5ms, and the response latencies of 96% of routing requests are lower than 10ms. However, some response latencies are still larger than 20ms. That is, Fig. 1(b) shows that there still exists a long-tail distribution under multiple ONOS controllers.

B. Analysis About the Long-Tail Latencies

Fig. 1(a) results from that the network only employs one controller. Due to the limited capability of the single controller, a large amount of requests have to queue in the controller. Therefore, there is a long-tail latency of routig response caused by the long queueing delays. In Fig. 1(b), the number of routing requests that each switch generates is different, even though they obey the same Poisson distribution. The skew-flow requests will make that some controllers are overload, but other controllers would be underutilized. As a consequence, there will be a long-tail latency. We illustrate the problem in Fig. 2. Two controllers are assigned to four switches and have the same processing time of 4ms. Assume $switch_1$ and $switch_2$ receive 4 requests each and that $switch_3$ and $switch_4$ receive 2 requests each. The requests received by $switch_1$ and $switch_2$ can only be processed by $controller_1$, which is assigned to manage them. This leads to a maximum latency of 32ms, but a load-aware selection obtains a maximum latency of 24ms. Fig. 2 shows that a quantity-based assignment strategy leads to long-tail latencies because it fails to accommodate the skew-flow requests.

Quantity-based allocation strategy is commonly employed by many controllers to balance the loads of controllers, such as ONOS [14]. That is a controller-to-switch assignment mechanism, which is abbreviated as the assignment mechanism of controllers. In this case, controllers coordinate to manage switches, and each controller manages an approximately equal quantity of switches. When deploying multiple instances of ONOS in a SDN, the bursty flows from a switch are sent to the same controller. Consequentially, a large number of requests have to queue in the controller. These queueing requests tend to incur long response latencies. However, those controllers, which do not receive bursty routing requests, may even be underloaded. In conclusion, the assignment mechanism fails to efficiently reduce the tail latencies of responses.

III. Framework of Controller Selection Mechanism

To overcome the drawback of assignment mechanism, we design a switch-to-controller selection scheme, which is abbreviated as the selection scheme of controllers. The selection scheme moves partial intelligences of the network to switches and relieves the loads of controllers. Meanwhile, the selection scheme can efficiently reduce tail latencies of responses.

A. Overview of Controller Selection Mechanism

For existing designs of control plane, the assignment mechanism of controllers is a static binding between switches and controllers, which fails to deal with the bursty and skew routing requests, and further incurs long response delays. To reduce the tail latencies of responses, routing requests from the same switch need to be processed by appropriate controllers. This means that the static binding between switches and controllers needs to be decoupled. A better mechanism is to enable switches to select controllers for routing requests.

For the assignment mechanism, the load balance among controllers means that each controller manages the same amount of switches. The load is denoted by the number of switches. Moreover, the load balance among controllers requires the coordination of controllers. The coordination will further aggravate the computing and communication overhead. However, in this paper, the selection scheme of controllers achieves the mapping between switches and controllers through switches conduct simple and actively probing.

IEEE Proof

Therefore, the selection scheme relieves the overhead of coordination and assignment in controllers.

In SDN, the controller periodically probes the switch state, and switches would respond to those probes and forward any switch notifications (e.g., link failures or recovery) to all live controllers [15]. When a switch suffers a failure scenario, it fails to respond to the probe. After that, the controller promptly updates its local topology state and replicates that update to all other controllers in the cluster. In addition, there have been lots of researches on how to achieve the consistency among distributed controllers [15]. These techniques are complementary to our approach.

We design the controller selection mechanism keeping in mind these four goals:

1) Light-weight: A light-weight probing method is needed to save the bandwidth of the secure channel. The probing for the controller selection uses the secure channel, which is the communication channel between the control plane and the data plane. The bandwidth of the channel can affect the performance of the whole network.
2) Scalable: The selection scheme of controllers should be irrelevant to the increasing number of controllers. The expansion of network size is common. The method of probing controller should accommodate the increase of deployed controllers and avoid to incur the communication overhead and the computing overhead.
3) Burst-immunity: The selection scheme should be burst-immune. There are bursty and skew-flow requests, which can lead to long response delays. To shorten the tail latency of responses, the selection scheme needs to accommodate the bursty and skew-flow requests.
4) Adaptive: The selection scheme of controllers should be adaptive. The capabilities of controllers may be heterogeneous and time-varying. To deal with the general situation, the selection scheme must cope and quickly react to heterogeneous and time-varying processing capabilities across controllers.

B. Load-Aware Selection Scheme of Controllers

Accommodating skew-flow requests across controllers necessitates a selection strategy of controllers. The strategy can make switches select faster controllers for routing requests. The controller with fewer unfinished requests can respond to routing requests faster when controllers have the same processing capability. In this paper, we first present a load-aware selection scheme. To realize this framework, the selection strategy needs to take into account the loads across multiple controllers in the network. The load means the number of unfinished requests. Our load-aware selection scheme is to select a controller with the fewest unfinished requests for newly generated requests.

Under the load-aware selection scheme of controllers, the switch needs to send a probing request to each controller after a switch receives a new flow. When the controller receives the probing request, it will return the number of unfinished requests to the switch. After the switch receives all probing results, it then sends the new routing request to the controller with the lightest load since that controller can respond the routing request fastest. The load-aware selection scheme aims to reduce tail latencies of responses by selecting the controller with the lightest load.

In Fig. 2, the load-aware selection scheme will work as follows. When a switch receives a new flow, it first probes the loads of controllers A and B. Based on the loads of controllers A and B, the switch sends the routing request to the controller with lightest load. This scheme can balance the loads of controllers A and B and can achieve better selection. When the controller finishes processing the routing request, it inserts the flow rules to related switches. Lastly, those switches will deal with the flow according to the actions of matched flow rules. In addition, the new arrival routing requests need to queue in controllers, when controllers are busy. However, the infinite length of queue will incur infinite response delays of routing requests. To cut the tail latency of response, it is necessary to ensure that the length of queue is finite in each controller.

C. The Condition to Finite Queue Length

We give the condition to achieve that the expected number of requests in per controller remains finite for all time. Consider the following model: requests arrive as a Poisson stream of rate λ at each switch. Requests are processed according to the first-in first-out (FIFO) protocol by controllers. The processing time for a request is exponentially distributed with mean μ. When there are m switches and n controllers in a network, requests arrive as a Poisson stream of rate $\frac{\lambda m}{n}$ at each controller. We obtain the following theorem. Note that $\frac{\lambda m}{n} < \mu$, and then the system will be stable, which means that the expected number of requests per controller remains finite in equilibrium. Theorem 1 shows that the system is stable for every $\frac{\lambda m}{\mu n} < 1$; that is, the expected number of requests in each controller remains finite for all time.

Theorem 1: The system is stable for every $\frac{\lambda m}{\mu n} < 1$; that is, the expected number of requests in each controller remains finite for all time.

Proof: When we treat all controllers as a whole and all switches as a whole, then the system can be seen as a M/M/1 system with Poisson arrival rate λm and average service rate μn. Let $P_k(t)$ denote the probability of that there are k requests in the whole system in time t. Accordingly, $P_{k+1}(t)$ denotes the probability where $k+1$ requests exist in the system in time t, and $P_k(t+\triangle t)$ denotes the probability of that there are k requests in the whole system in time $t+\triangle t$. Now we consider the evolution of the system. In the time $[t, t+\triangle t]$, the process of evolution has some attributes just as follows:

- One request comes with the probability $\lambda m \triangle t$, and the probability of no request comes is $1-\lambda m \triangle t$.
- One request departures with the probability $\mu n \triangle t$, and the probability of no request departure is $1-\mu n \triangle t$.
- The situation of more than one request comes and departures in $\triangle t$ is a small probability event, and it can be ignored.

There are 4 types of evolution process, and we list them in Table I. Take type B for an example. Since k is the number

IEEE Proof

TABLE I
FOUR TYPES OF EVOLUTION PROCESS

Type	state in time t	come	departure	state in time $(t+\triangle t)$
(A)	k	0	0	k
(B)	$k+1$	0	1	k
(C)	$k-1$	1	0	k
(D)	k	1	1	k

of requests in the whole system, so $k+1$ means that there are $k+1$ requests in the system in time t. After that, when a request departs during $\triangle t$, there would be k requests in the system in time $t+\triangle t$. Accordingly, we can get the possibility of type A is $P_k(t)(1-\lambda m\triangle t)(1-\mu n\triangle t)$, possibility of type B is $P_{k+1}(t)(1-\lambda m\triangle t)\mu n\triangle t$, possibility of type C is $P_{k-1}(t)\lambda m\triangle t(1-\mu n\triangle t)$ and possibility of type D is $P_k(t)\lambda m\triangle t\mu n\triangle t$. Then, $P_k(t+\triangle t)$ should be the sum of all 4 types, shown in Equation (1).

$$P_k(t+\triangle t) = P_k(t)(1-\lambda m\triangle t-\mu n\triangle t) + P_{k+1}(t)\mu n\triangle t + P_{k-1}(t)\lambda m\triangle t + o(\triangle t) \quad (1)$$

And let $\triangle t \to 0$, we can get a differential equation, shown in Equation (2).

$$\frac{dP_k(t)}{dt} = \lambda m P_{k-1}(t) + \mu n P_{k+1}(t) - (\lambda m+\mu n)P_k(t) \quad k=1,2,\ldots \quad (2)$$

Noted that if $k=0$, there will exist only type A and type B, shown in Equations (3) and (4).

$$P_0(t+\triangle t) = P_0(t)(1-\lambda m\triangle t) + P_1(t)(1-\lambda m\triangle t)\mu n\triangle t \quad (3)$$

$$\frac{dP_0(t)}{dt} = -\lambda m P_0(t) + \mu n P_1(t) \quad (4)$$

We just have interest in the equilibrium point, and the derivative is 0 in fixed point. Thus, we get Equation (5).

$$\begin{cases} -\lambda m P_0 + \mu n P_1 = 0 \\ \lambda m P_{k-1} + \mu n P_{k+1} - (\lambda m+\mu n)P_k = 0 \quad k \geq 1 \end{cases} \quad (5)$$

Resolve equation (5), we can get $P_k = (\lambda m/\mu n)^k P_0$. If $\frac{\lambda m}{\mu n} < 1$, then the sequence P_k will be decrease. And we know probability is non-negative, that means $P_k \geq 0$. If a sequence is bounded and monotone, it converges [16]. So there exist K, when $k > K$, $P_k = 0$. Then the expected total number of requests in all controllers remains finite. ■

Theorem 1 shows that the expected total number of requests in each controller remains finite, when $\frac{\lambda m}{\mu n} < 1$. Therefore, to achieve the finite queue length, it is essential to ensure that $\frac{\lambda m}{\mu n} < 1$. When the network size m increases, if $\frac{\lambda m}{\mu n} \geq 1$, it is necessary to increase the number of deployed controllers n, otherwise, the queue length of some controllers will be infinite, and that will incur infinite response delays. Another method to limit the queue length of controllers is to drop some routing requests, which can limit the value of λ and achieve $\frac{\lambda m}{\mu n} < 1$.

D. Limitations of Load-Aware Selection Scheme

Fig. 3 shows the Cumulative Distribution Function (CDF) of response latencies of routing requests under different schemes.

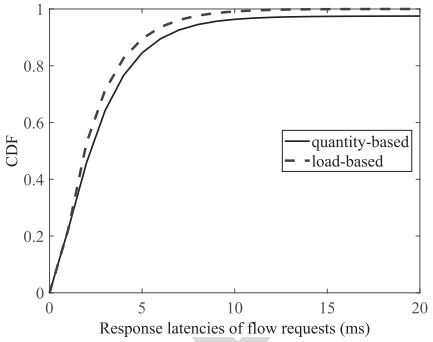


Fig. 3. Response latencies of routing requests under different schemes.

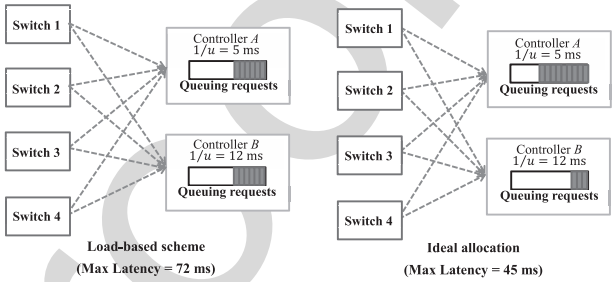


Fig. 4. Distinct selection schemes incur different response latencies.

In Fig. 3, the response latencies of 94% of routing requests are lower than 5ms after adopting the load-aware selection scheme. However, for the quantity-based assignment, only 86% of routing requests can be responded in 5ms. In Fig. 3, all routing requests can be responded in 10ms under the load-aware scheme. Therefore, Fig. 3 indicates that our load-aware selection scheme can reduce the tail latency of responses than the prior quantity-based allocation method when controllers are homogeneous and exhibit the same processing capabilities. However, we can see that both curves (Quantity-based and Load-aware) are very close to each other, which means that the load-aware selection strategy narrowly reduce the long-tail latency. Furthermore, the load-aware selection scheme faces three challenges. First, controllers are heterogeneous. Second, the processing capabilities of controllers are dynamically changing. Third, the cost of probing is too huge. Therefore, the load-aware selection scheme is still insufficient to completely cut the tail latency.

Fig. 4 plots an illustrative example of the limitation. For two controllers, the processing time per request in controller *A* and controller *B* are 5ms and 12ms, respectively. Assume all four switches receive a burst of 3 requests each. Each request needs to be forwarded to a single controller. If every switch selects a controller using the load-aware scheme, it will result in each controller receiving an equal share of the requests. This leads to a maximum latency of 72ms, whereas an ideal selection in this case obtains a maximum latency of 45ms. We note that the load-aware scheme will prefer faster controllers over time, but purely relies on the load information. Therefore, when controllers are heterogeneous, the load-aware selection scheme can not efficiently shorten the tail latency.

Controllers are commonly heterogeneous for primarily three reasons. First, the hardware is heterogeneous. Controllers run

IEEE Proof

in commercial servers. These servers can be heterogeneous due to different hardware configurations, such as CPU and memory. Second, the software is heterogenous. There are multiple different controllers developed by different organizations [10], such as NOX, Beacon, Floodlight, Ryu, ONOS and OpenDaylight, etc. Those controllers themselves have different performances. Third, the function is heterogeneous. There are some management applications running in controllers for achieving different functions [12], [17], and these applications will consume some resources of controllers. Consequentially, controllers have different remaining capabilities for processing routing requests, even if the controllers run in servers with the same setting. In this case, queueing routing requests in controllers with low processing capabilities will lead to long response latencies.

Additionally, the load-aware scheme probes the loads of all controllers, and then selects the controller that has the lightest load. However, this probing will incur the overhead of communication and aggravate the loads of controllers when there are a large number of controllers in a large-scale network. For example, there are m switches and n controllers in a network. Suppose that each switch receives λ routing requests in 1ms. There are $2\lambda \times m \times n$ times communications between switches and controllers during 1ms. Meanwhile, each controller needs to evaluate its own load λm times in 1ms. The cost of probing is too huge for the load-aware selection scheme.

The load-aware controller selection scheme can only reduce tail latencies of responses under the homogeneous controllers. However, the network environment is time-varying in real situations, not only in the processing capabilities of controllers but also in the number of routing requests from switches. We further propose a delay-aware selection scheme of controllers, which can adapt to the variations of the network environment.

IV. Delay-Aware Selection Scheme of Controllers

We design the delay-aware selection scheme of controllers while keeping the design goals of controller selection mechanism in mind. We first show an overview of the delay-aware selection scheme. Then, we present two major components of the delay-aware selection scheme, the selection models of controllers and the queue management mechanism of controller.

A. Overview of Delay-Aware Selection Scheme

To address those problems faced by the load-aware selection scheme, we further design the delay-aware selection scheme of controllers, which is adaptive to the heterogeneous controllers as well as to the dynamic behaviours of flows. The delay-aware selection scheme needs to probe the response delays of controllers for routing requests and send the routing requests to the controller with the smallest response delay. The latency of routing response denotes the time interval from sending the routing request to receiving the flow rules generated by the controller and is composed of the queueing delay and the processing delay, as shown in Definition 1. The response delay is an approximate evaluation of response time. Through probing response delays, the delay-aware selection scheme can accommodate the heterogeneous controllers, while fewer routing requests will be sent to the controllers with low processing capabilities.

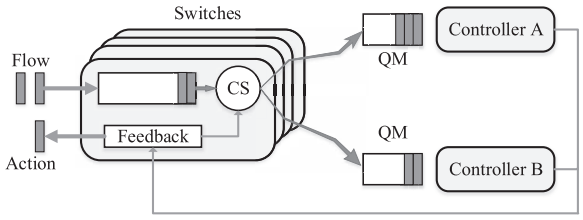


Fig. 5. Overview of controller selection scheme. CS: Controller Selection scheduler, QM: Queue Management of controller.

Definition 1: The latency of routing response denotes the time interval from sending the routing request to receiving the flow rules generated by the controller.

Furthermore, the capabilities of controllers are time-varying. With the development of SDN, there are more and more applications running in controllers. When switches send vast requests to the controller that has fast capability of response at before, a large number of requests have to queue in controllers if the capabilities of controllers decrease due to other applications' overconsumption of resources.

Our delay-aware selection scheme includes two major components, controller selection (CS) and queue management (QM). Recall the design goal of the selection scheme in Section III-A, CS can achieve that the selection scheme is light-weight, scalable and burst-immune, and QM achieves the goal of adaptivity. First, we design a selection scheme of controllers, which can select the controller based on a little feedback from the controllers, and thus, is light-weight. Second, instead of probing all controllers, the switch randomly probes d controllers where $d \geq 1$. The probing is scalable and independent of the network size. Third, the selection scheme can make switches conduct once controller selection for each flow or a batch of flows. It can make those requests be processed by different controllers, and thus can avoid the influence of the bursty and skew-flow requests. Last, through estimating response delays of routing requests, switches can send routing requests to the controller that has the smallest response delay. Based on this estimation, the selection of controllers can be adapted to the heterogeneous and time-varying processing capabilities.

Fig. 5 depicts the framework of the adaptive switch-to-controller selection scheme. When a request is issued at a switch, the switch will work based on Algorithm 1. The switch randomly probes d controllers, where $d \geq 1$. The d controllers then estimate their response delays for the routing request based on Algorithm 2 and return the response delays ψ to the switch. If ψ_i of controller i exceeds the max response delay RD_{max} limit, then controller i will return the response delay $\psi_i = -1$. When the switch receives response delays of d controllers, it will select the controller that has the smallest response delay. If all response delays are lower than 0, the switch will reselect a controller whose queue length does

IEEE Proof

Algorithm 1 The Selection of Controllers

Require: Controller set C, d.
randomly probe d controllers from C;
send estimating request to the d controllers;
$\psi \leftarrow$ response delays of d controllers;
if there exists $\psi_x \geq 0$ **then**
  $id \leftarrow arg\ min\{\psi_x \geq 0\}$;
else
  for $i = 1$ to C.length **do**
    send estimating request to controllers $C[i]$;
    $\psi_0 \leftarrow$ response time of $C[i]$;
    if $\psi_0 >= 0$ **then**
      $id = i$;
      break;
send the routing request to controller $C[id]$.

Algorithm 2 Queue Management of Controller

Require: the max response delay, RD_{max}.
receive an estimating request from switch s;
calculate the average processing time of requests $\bar{v}$;
if $r_i < \gamma_i$ **then**
  $\psi_i \leftarrow \frac{r_i}{\gamma_i}\bar{v}$;
else
  $\psi_i \leftarrow C[q_i mod \gamma_i] + (\lfloor q_i/\gamma_i \rfloor + 1) \times \bar{v}$;
if $\psi_i > RD_{max}$ **then**
  $\psi_i = -1$;
send ψ_i to switch s.

not exceed limit for the routing request. Last, the switch will send the request to the selected controller.

B. The Selection Models of Controllers

When there are only a few controllers in the network, it is feasible to probe all controllers. Considering that this probing could occupy extra bandwidth of the secure link, it is essential to design a per-flow light-weight probing method.

1) Active Per-Flow Selection of Controllers: To deal with the skew-flow requests and reduce response tail latencies, the switches need to select controllers for each flow. Instead of the controller-to-switch assignment, the active per-flow selection of controllers makes that the routing requests from the same switch can be processed by different controllers. This selection scheme can fully exploit the capabilities of controllers and efficiently reduce response tail latencies.

To reduce the bandwidth consumption of probing controllers, one method is to reduce the number of probed controllers. There is a tradeoff between response tail latencies and the number of probed controllers. Probing more controllers can achieve fewer response tail latencies. However, that also means more bandwidth consumption and computing overhead in controllers. The number of controllers increases as the network scale grows. In this case, checking all controllers has a huge cost. To achieve the light-weight probing, we randomly probe d controllers instead of checking all controllers, where $d \geq 1$. Furthermore, Azar *et al.* [18] have shown that having just two random choices (i.e., $d = 2$) yields a large reduction in the maximum load over having one choice. This method has been widely studied and applied [19]. Inspired by this fact, our active per-flow selection is to probe two controllers and is thus scalable. Meanwhile, the active per-flow selection of controllers can efficiently reduce the bandwidth consumption of the secure link and the computing load of controllers.

Instead of probing the loads of the controllers, probing response delays of controllers can better reduce response tail latencies. The probing of response delays requires that these controllers evaluate their own response delays for routing requests. Since the selection of controllers only needs to get a numerical value of response delay, the selection of controllers is light-weight. Moreover, the heterogeneous and time-varying processing capabilities of controllers increase the complexity of evaluation for response delays of controllers. The response delay estimate model will be introduced in Section IV-C.

2) Active Selection of Controllers for a Batch of Flows: When switches meet bursty-flow requests or when the arrival of routing requests is frequent, conducting a controller selection for each request still aggravates the bandwidth consumption and the loads of controllers even if only two controllers need to be probed in one controller selection. Conducting the controller selection for a batch of arrival routing requests is needed to increase the scalability of the controller selection mechanism, when switches suffer bursty flows.

For active selection of controllers for a batch of flows, the switch conducts one controller selection after it receives the first flow request. When we set the batch size as δ, it means that the following $\delta - 1$ requests will be sent to the same controller with the first request. That is, the result of controller selection for the first request will be shared by δ requests. In addition, the batch selection is irrelevant to the rate of requests because δ denotes the number of requests. Therefore, although there would be the high rate of requests in the beginning when the switch changes its controller, those requests would not be sent to the same controller. Given that the request arrival process at each switch is a Poisson process with rate λ, the arrival duration for a flow is exponentially distributed with mean $1/\lambda$. Therefore, the arrival duration of δ flows is also exponentially distributed with mean δ/λ.

There is a tradeoff between the rounds of controller selection and the performance of controller selection. If δ is too small, it is obvious that controller selection should be frequently conducted. However, it will decrease the performance of controller selection when δ is too large. It is worth noting that it is unnecessary to adopt the batch selection when the arrival of flows is scattered.

C. Queue Management Mechanism of Controller

The queue management of controller makes it so that the selection of controllers can cope and quickly react to heterogeneous and time-varying processing time across controllers. Our queue management mechanism of controller includes response delay estimate and queue length bound.

1) Response Delay Estimate Model: As depicted in Section III-D, the load-aware selection scheme of controllers

IEEE Proof

can not accommodate the heterogeneity of controller. To efficiently reduce the long-tail latency of routing response, switches should select controllers with lower response delays for each routing request.

Request response time consists of the queueing time and the processing time. Furthermore, the queueing time is related to the length of queue, which is equal to the number of queueing requests. Meanwhile, to estimate the queueing time, it is essential to estimate the processing time of each request. In our design, the controller records ν_j, which is the processing time of the latest responded jth requests. Given the number of the latest finished requests s, we calculate $\bar{\nu}$, which denotes the average processing time of s requests in controller i. Thus, $\bar{\nu} = \frac{1}{s}\sum_{j=1}^{s}\nu_j$. We use $\bar{\nu}$ to estimate the processing time of requests.

Consider that the controller can process multiple routing requests simultaneously. We use γ_i to denote the number of requests that $controller_i$ can simultaneously process. q_i and r_i are the number of queueing and running requests in controller i, respectively. To improve the system utilization, the controller that has idle running slots should have a lower estimated response delay. Therefore, the estimated response delay $\psi_i = \frac{r_i}{\gamma_i}\bar{\nu}$ when $r_i < \gamma_i$. When there are requests queueing in a controller, the controller records the running duration $A[k]$ of the running request in the kth slot where $1 \le k \le \gamma_i$. The controller then estimates that the queueing request will run in which slot. To achieve this goal, we use $B[k] = |\bar{\nu} - A[k]|$ and then sort $B[k]$ as non-decreasing order. Then, the controller estimates that the request will run in $[(q_i \mod \gamma_i) + 1]$th slot. We can get the queueing time $wt_i = B[(q_i \bmod \gamma_i)+1]+(\lfloor q_i/\gamma_i \rfloor)\times\bar{\nu}$. At last, $\psi_i = wt_i + \bar{\nu}$ when $r_i = \gamma_i$.

In summary, $controller_i$ uses the following estimation function for response delay:

$$\psi_i = \begin{cases} B[(q_i mod \gamma_i) + 1] + (\lfloor q_i/\gamma_i \rfloor + 1) \times \bar{\nu} & r_i = \gamma_i \\ \frac{r_i}{\gamma_i}\bar{\nu} & r_i < \gamma_i \end{cases} \quad (6)$$

When a switch sends an estimating request to $controller_i$, the $controller_i$ adopts the formula (6) to estimate the response delay. In general, $\gamma_i = 1$ means that the controller only can process one request once. In this case, $\psi_i = B[1]+(q_i+1)\times\bar{\nu}$. We suppose that the controller is empty at the beginning. After that, $\bar{\nu}$ is equal to the average processing time of finished requests when the number of finished messages is lower than the given threshold s.

2) Cutting Tail Latencies: Since switches conduct controller selection simultaneously, there may be "herd behaviors," wherein multiple switches are coaxed to direct requests towards the best controller. There are many requests queueing in a controller under herd behavior that could leads to long-tail latencies of routing responses. Moreover, it is possible that the probed controllers all have low processing capabilities or long queues. In this case, it is not suitable to select a controller from those probed controllers.

To cut long-tail latencies and reduce the influence of herd behavior, the controller necessitates to bound its queue length. Determining the length of queues at controllers is crucial. Queues that are too short lead to lower controller utilization, as resources may remain idle between allocations. Queues that are too long may incur excessive queuing delays.

When fewer requests are sent to a controller, this may incur under-utilization of its resources, whereas significant delays may occur when requests need longer processing time. Hence, after estimating request response delay, we further design a delay-aware bounding mechanism to bound queue length, which can accommodate the heterogeneous and time-varying capabilities of controllers. Meanwhile, bounding queue length can efficiently weaken the influence of herd behaviors. At one point, a controller receives a burst of flows, and that exceeds the limit of queue length. After that, the following flows will not queue in the controller until the queue length is lower than the limit. This delay-aware bounding mechanism relies on the response delay estimation of request, which is reported by the controller.

In particular, we specify the maximum response delay RD_{max} that a request is allowed to wait in a queue. When we are about to place a request at the queue of $controller_i$, we first check the estimated response delay ψ_i reported by $controller_i$. Only if $\psi_i < RD_{max}$ is the request queued at that controller. We sample d controllers while conducting the controller selection. If the d selected controllers all do not satisfy the maximum response delay constraint. The switch needs to reselect a new controller. Using this method, the number of requests in each controller gets dynamically adapted based on the current capability of the controller.

RD_{max} is set to make requests prefer faster controllers. Furthermore, RD_{max} can be dynamically regulated to fit the variation of controllers' capabilities. For controllers that have low processing capabilities, RD_{max} can limit the number of queueing requests in these controllers. After that, these requests can be sent to faster controllers. However, if RD_{max} is too small, most of controllers refuse to receive new requests because their response delays exceed the limit of RD_{max}. In this case, it is necessary to extend the value of RD_{max}.

V. Performance Evaluation

We start with the evaluation methodology and scenarios. In this section, we evaluate the selection schemes of controller and the assignment mechanism of controller under the general settings of controllers, the queue length bound RD_{max}, the heavy request-skews, the time-varying service rates and the batch selection.

A. Experimental Setup

We build a discrete-event simulator wherein workload generators create flow requests at a set of switches. These switches then employ the controller selection scheme to select a controller for each request. Unless otherwise specified, the network consists of 300 switches and 40 controllers. However, when a large amount of flow requests emerges, we need to employ more controllers to deal with the burstly flows. The workload generators create flow requests according to a Poisson arrival process to mimic arrival of user requests at web servers [20]. Unless otherwise specified, the Poisson

IEEE Proof

arrival process is with $\lambda = 0.5$ during 1ms. At the beginning, the system is empty, and there are no requests. With the system running, the switches start to produce flow requests and select controllers for flow requests. We run the system 1000ms, and the number of arrival flow requests is about 150, 000. Each controller maintains a FIFO request queue. Moreover, in the settings of controllers, each controller can service a tunable number of requests in parallel (10 in our settings). The processing time each request experiences is drawn from an exponential distribution (as in [21]) with a mean processing time $\mu^{-1} = 2$ms. Furthermore, we incorporate controller heterogeneity into the network as follows: each controller, independently and with a uniform probability, sets its service rate to a different μ, where $\mu = 0.5$ or $\mu = 0.1$ in our settings. To estimate the response delay of each request, we set $s = 100$, That is, $\bar{v}$ denotes the average processing time of the latest finished 100 requests. We repeat every experiment 20 times using different random seeds, and then get the average result.

We compare our design against three strategies:

1) **Quantity-based allocation**: Controllers achieve load balance through balancing the number of switches, which each controller manages. Currently, ONOS controller utilizes this strategy to balance the loads of distributed controllers.
2) **Load-aware selection**: The switch probes two controllers for each request and sends the request to the controller with fewer number of requests because probing all controllers is not scalable.
3) **Delay-aware selection**: The switch probes two controllers for each request and gets request response delays, which rely on the feedbacks of probed controllers. Then, the switch sends the request to the controller with smaller response delay.

B. The Impact of d

Azar *et al.* [18] have shown that the situation of $d = 2$ yields a large reduction in the maximum load over $d = 1$, while each additional choice beyond two decreases the maximum load by just a constant factor. Further, to verify the theory and determine the value of d, we evaluate the impact of d on the performance of the delay-aware selection strategy under heterogeneous controllers where d denotes the number of probed controllers. The processing time of each request in each controller is drawn from an exponential distribution where μ was randomly set as 0.5 or 0.1. Other parameters are the same as Section V-A.

Fig. 6 shows the delay-aware selection strategy significantly reduces the mean response time as the value of d increase from 1 to 2. However, probing more controllers just incur a little bit reduction on the mean response time after d exceeds 2. In addition, multiple switches may simultaneously select the same controller if each of them randomly probes more controllers. This would in turn aggravates the load of the controller. Thus, we set $d = 2$ in the next experiments.

C. General Settings of Controllers

We evaluate the performances of different schemes with heterogeneous controllers. We employ 60 controllers where the

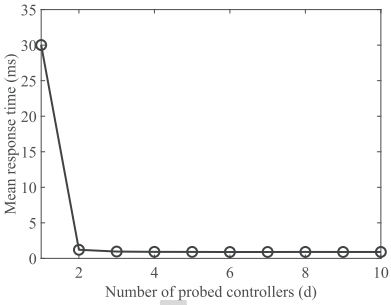
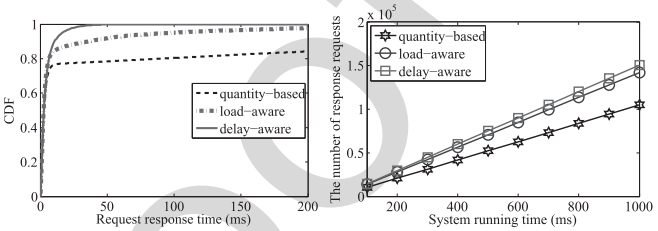


Fig. 6. The impact of d on the performance of the delay-aware selection strategy.



(a) Request response time with different (b) Throughput under different schemes.

Fig. 7. The performances of different schemes where controllers are heterogeneous.

bound of maximal response delay $RD_{max} = 20$ms. The other settings of experiments are consistent with that of Section V-B.

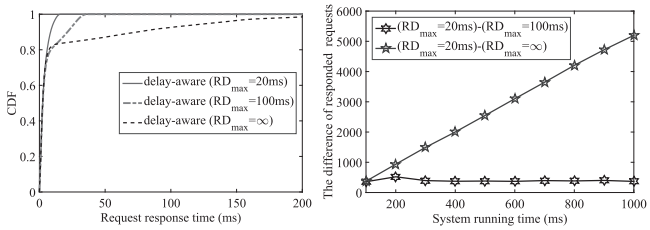
Fig. 7(a) shows that our delay-aware scheme can significantly reduce the response tail latencies. Basically, all requests can be finished in 50ms while adopting our delay-aware scheme. The load-aware scheme achieves better performance than the quantity-based scheme in Fig. 7(a). Over 90% of requests can be processed during 150ms based on the load-aware scheme. The quantity-based scheme leads to long response delays, and there are over 20% of requests whose response delays are more than 200ms in Fig. 7(a). This is because a large of requests queue in controllers that have lower processing capabilities. Meanwhile, the load-aware scheme also failed to respond to requests quickly. The processing delays of flow requests in different controllers are different when controllers are heterogeneous. As a consequence, selecting a controller by the number of requests is not efficient. Fig. 7(a) reveals that our delay-aware scheme achieved the lowest response duration due to not only estimating response delay but also cutting tail latencies. Fig. 7(a) also reveals that our delay-aware scheme can be adapted to the system where controllers have heterogeneous capabilities.

Fig. 7(b) shows that our delay-aware scheme can respond to more requests than the load-aware scheme and the quantity-based scheme can in the same time period. Meanwhile, the throughput difference among schemes grows as the system runs. In summary, with heterogeneous controllers, our delay-aware scheme can efficiently reduce response tail latencies and improve the throughput of controllers.

D. Impact of Queue Length Bound RD_{max}

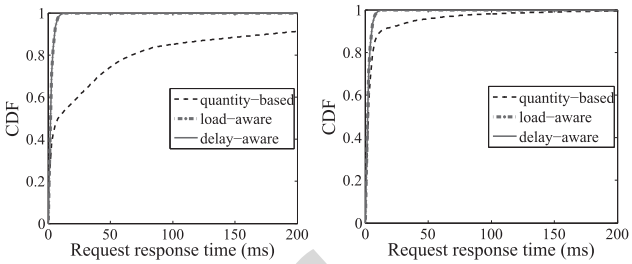
We evaluate the impact of RD_{max} on the performance of the delay-aware scheme. We set $RD_{max} = 20$ms, $RD_{max} = 100$ms and $RD_{max} = \infty$ respectively. $RD_{max} = \infty$ means

IEEE Proof



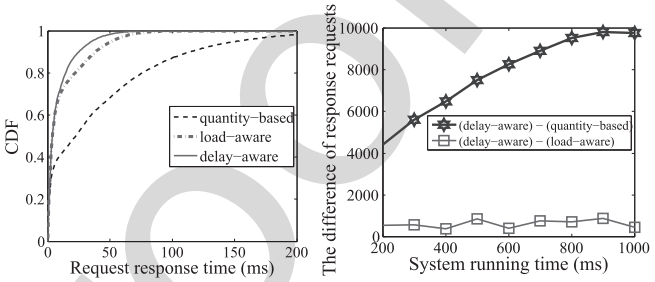
(a) The performance of delay-aware scheme under different RD_{max} settings.(b) The difference of responded requests under different RD_{max} settings.

Fig. 8. The impact of RD_{max} on the performance and throughput of the delay-aware scheme.



(a) $Request\text{-}skew$=20%. (b) $Request\text{-}skew$=50%.

Fig. 9. Request response time with different schemes under the heavy request-skews.



(a) Request response time with different schemes. (b) The difference of responded requests between different schemes.

Fig. 10. The performances of different schemes under the time-varying service rates.

that there is no limit on the queue length of controllers. Other parameters are the same with Section V-C.

Fig. 8(a) reveals that our delay-aware scheme has better performance when RD_{max} has a smaller value. Under $RD_{max} = 20$ms, all requests can be finished in 20ms. However, The maximal response delay is 40ms under $RD_{max} = 100$ms. Therefore, the performance of delay-aware scheme under $RD_{max} = 20$ms is better than that of $RD_{max} = 100$ms. Comparing with $RD_{max} = \infty$, delay-aware scheme can significantly reduce response tail latencies when $RD_{max} = 20$ms. Furthermore, we compare the throughput of controllers under different RD_{max} settings. Fig. 8(b) shows that the system can respond to 300 more requests under $RD_{max} = 20$ms than $RD_{max} = 100$ms. It was obvious that controllers have higher throughput when $RD_{max} = 20$ms than when $RD_{max} = 100$ms and $RD_{max} = \infty$. The difference of responded requests between $RD_{max} = 20$ms and $RD_{max} = 100$ms remains stable. However, the difference of responded requests between $RD_{max} = 20$ms and $RD_{max} = \infty$ grows as the system runs.

Fig. 8 reveals that the setting of RD_{max} can make flow requests prefer the controllers that have faster processing capabilities. Additionally, it is noteworthy that RD_{max} can not be set too small, otherwise, there are no available controllers while conducting the controller selection.

E. Performance Under Heavy Request-Skews

In this section, we study the effect of heavy demand skews on the observed latencies where controllers are homogeneous with average server rate $\mu = 0.5$. We set request-skew= 20% and request-skew= 50%. That is, 20% and 50% of switches generated 80% of the total requests towards the controllers. Most of parameters are inherited from Section V-A. To enable 20% of switches to generate 80% of the total requests, we randomly select 60 switches and set the arrival rate of flow requests $\lambda = 2$. Other switches set $\lambda = 0.125$. Under request-skew= 50%, half of the switches set $\lambda = 0.8$, and the other half of the switches set $\lambda = 0.2$. We set $RD_{max} = 150$ms because there are too many requests in a short time and these requests have to queue in controllers.

Fig. 9(a) shows that over 5% requests have response delays of more than 200ms for the quantity-based scheme. The quantity-based scheme suffers decreased performance due to the request-skews. However, the load-aware and delay-aware schemes can significantly reduce response tail latencies. Based on the load-aware and delay-aware schemes, all requests can be finished in 10ms in Fig. 9(a). Fig. 9 reveals that the load-aware and delay-aware schemes achieve very similar performances since controllers are homogeneous in this section. Meanwhile, the quantity-based scheme suffers decreased performance due to the request-skews. Under request-skews, a part of switches generate a large number of the requests, which incur long queues in some controllers for the quantity-based scheme.

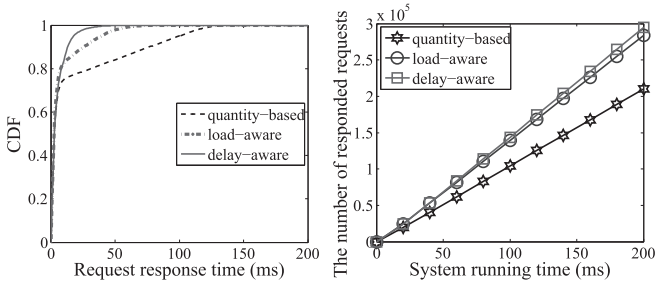
Comparing Fig. 9(a) and Fig. 9(b), we can find that the quantity-based scheme under request-skew= 50% achieves a lower response latency than under request-skew= 20%. It is because the load balance among controllers where request-skew= 50% is better than that of request-skew= 20%. Moreover, Fig. 9 also reveals that our delay-aware scheme can accommodate the heavy request-skews.

F. Impact of Time-Varying Service Rates

In this section, we study the effect of the service rate fluctuation on the tail latency of response. We change the average service rates of controllers in the system every 50ms, and all controllers randomly set $\mu = 0.5$ or $\mu = 0.1$. Other parameters are inherited from Section V-C.

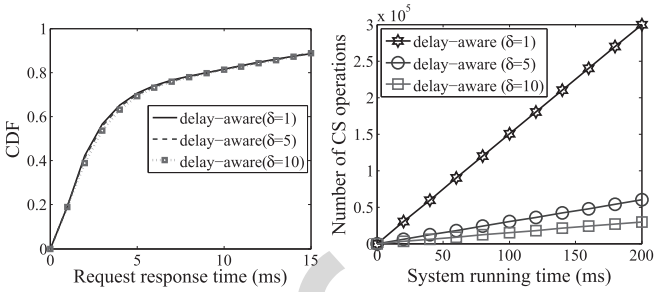
Fig. 10(a) reveals that our delay-aware scheme can respond to all requests in 60ms, the load-aware scheme can respond to all requests during 80ms, and the response tail latency of the quantity-based scheme is more than 200ms. Therefore, our delay-aware scheme can efficiently reduce response tail latencies. For the quantity-based scheme, it could not exploit the feedbacks of controllers to select the controller and further

IEEE Proof



(a) Request response time with different schemes. (b) Throughput under different schemes.

Fig. 11. The performances of different schemes under the batch selection.



(a) The performance of delay-aware scheme. (b) The number of CS operations. CS: the selection of controller.

Fig. 12. The impact of the batch size δ.

suffers lower system utilization. The load-based scheme also suffers lower performance because it fails to consider the time-varying service rate. Fig. 10(a) reveals that our delay-based scheme can accommodate time-varying service rate.

Fig. 10(b) shows that our delay-aware scheme can respond to more requests than the quantity-based scheme and the load-aware scheme. With the increase of system running time, the advantage of the delay-aware scheme is more obvious than the quantity-based scheme in Fig. 10(b). Meanwhile, the difference of responded requests between the delay-aware scheme and the load-aware scheme cyclically fluctuates because the service rates of controllers are periodically changed. In summary, our delay-aware scheme can be better adapted to the time-varying service rate and can efficiently reduce tail latencies of responses.

G. Evaluation for Batch Selection

In this section, we evaluate the performance of different schemes for batch arrival requests. We set the arrival rate of flow requests $\lambda = 5$. There are about $300,000$ requests during 200ms. To deal with the burst requests, we employ 150 controllers, and each controller can process 40 requests in parallel. We set the batch size $\delta = 10$ and $RD_{max} = 15$ms, and other parameters are set as Section V-C.

Fig. 11(a) reveals that our delay-aware scheme can still reduce tail latencies of responses under the batch selection. Our delay-aware scheme can respond to all requests in 40ms, and the load-aware scheme can respond to all requests during 60ms. However, the response tail latency of the quantity-based scheme is more than 130ms. The performance of the quantity-based scheme is mainly affected by controllers that have lower processing capabilities. The tail latency of response under the load-aware scheme is generated because it fails to accommodate the heterogeneous controllers. Meanwhile, Fig. 11(b) shows that our delay-aware scheme can respond to more flow requests than either of the other two schemes. This means that our delay-aware scheme can not only reduce the tail latency, but can also improve the throughput of controllers under the batch selection.

Impact of Batch Size δ: Furthermore, we evaluate the impact of the batch size δ on the performance of the delay-aware scheme. Fig. 12(a) depicts the performances of the delay-aware scheme under different batch sizes. Fig. 12(a) shows that the performance of the delay-aware scheme has a modest decrease with the increase of the batch size δ. When the batch size δ increases, it means that more requests will be sent to the same controller, even though the controller has a low processing capacity. As a consequence, the delay-aware scheme suffers a little of decreased performance. we can find that experiments show a similar performance under the different settings of δ in Fig. 12(a). Although more requests would be directed to the controller with a low processing capacity at some time when the value of δ is larger, after that, the following requests would have less chance to select the controller. Therefore, the decrease of performance is modest. Furthermore, Fig. 12(b) reveals that the number of controller selection operations dramatically decreases when the batch size δ goes up. The fewer controller selection operations in switch end will incur less bandwidth consumption in the secure links between switches and controllers and less computing load in controllers. In conclusion, active selection of controllers for a batch of flows can efficiently reduce the resourse consumption of communication and computing with a little bit of performance reduction.

VI. Related Work

A. Network Softwarization

Network softwarization is a transformation trend for designing, implementing, and managing the 5G and next generation networks. It exploits the benefits of software to enable the redesign of network and service architectures, optimize the expenditure and operational costs, and bring new values in the infrastructures. The key enablers consist of the network function virtualization (NFV), software-defined networking (SDN) and cloud computing, etc. Meanwhile, 5G systems will also rely on these technologies to attain the systems flexibility and elasticity [2].

Along with recent and ongoing advances in cloud computing, it has become promising to design flexible, scalable, and elastic 5G systems benefiting from advanced virtualization techniques of cloud computing [1]. Taleb and Ksentini [22] introduces the Follow-Me Cloud concept and proposes its framework. There has been research on the possibility of extending cloud computing beyond data centers toward the mobile end user, providing end-to-end mobile connectivity as a cloud service [23]. Software defined networking (SDN)

IEEE Proof

acts as a promising enabler for network softwarization and plays a crucial role in the design of 5G wireless networks [1]. SDN has been also utilized in the virtualization of mobile network functions [24]. Kempf *et al.* [25] describe an evolution of the mobile Evolved Packet Core (EPC) utilizing SDN that allows the EPC control plane to be moved into a data center. Taleb *et al.* [1] introduce the concept of "Anything as a Service" (ANYaaS), which relies on the reference ETSI NFV architecture to orchestrate and manage important services. NFV aims at offering network services using network functions implemented in software and deployed in an on-demand and elastic manner on the cloud [26]. Medhat *et al.* [5] introduce a service function chaining taxonomy as the basis for the subsequent state-of-the-art analysis.

B. SDN Scalability

To tackle the problem of scaling SDN, Xie *et al.* [27] resort to design the distributed controllers, such as Onix [28] and ONOS [14], which try to distribute the control plane while maintaining a logically centralized management. These approaches balance the load of controllers based on the number of switches, which can not efficiently reduce the tail latencies of responses. Aissioui *et al.* [29] propose a two-level hierarchical controller platform to address these scalability and performance issues in the context of 5G mobile networks.

One key limitation of the distributed controllers is that the mapping between a switch and a controller is statically configured. The static configuration results in the uneven load distribution among the controllers. Bari *et al.* [30] propose a management framework, which periodically evaluates the current controller-to-switch assignment. After that, it needs to decide whether to perform a reassignment. If a reassignment is performed, the management framework also changes the controller-to-switch assignment in the network. Dixit *et al.* [31] propose ElastiCon, an elastic distributed controller architecture in which the controller pool is dynamically grown or shrunk according to traffic conditions. In this case, the load is dynamically shifted across controllers, which similarly relieves the static mapping between a switch and a controller. Zhou *et al.* [32] propose a dynamic and adaptive algorithm (DALB), which is running as a module of SDN controller. The controllers in distributed environment can cooperate with each other to keep load balancing. Overloaded controller can be detected, and high-load switches mapped to this controller can be smoothly migrated to the under-load controllers. However, these dynamic frameworks require that the control plane monitors the state of the whole network and conducts the reassignments, which aggravate the computing load of the control plane.

In contrast to these works, our approach relies on simple and inexpensive feedback of controllers and efficiently relieve the load of the control plane. Mao and Shen [33] use the principles of SDN to achieve the server load balancing by setting the SDN flow table, which does not aim to solve the load balance of the distributed controllers of SDN. Palma *et al.* [34] develop the QueuePusher, which is a queue management extension to OpenFlow controllers supporting the Open vSwitch Database Management Protocol (OVSDB) standard. QueuePusher can generate the appropriate queue configuration messages for switches. In addition, there have been lots of researches on how to achieve the consistency among distributed controllers [15]. These techniques are complementary to our approach.

VII. Conclusion

NFV and SDN can dynamically redistribute the flow across appropriate VNFs or service function chains if the controller configures a desired routing path for each network flow resulting from NFV applications. In this paper, we present the long-tail observations of the routing response latencies while using the up-to-date controller-to-switch assignment mechanism. To tackle this essential problem, we first propose a light-weight and load-aware switch-to-controller selection scheme to cut the long-tail response latency under the simple scenario of homogeneous controllers, and then design a general delay-aware switch-to-controller selection scheme to fundamentally cut the long-tail response latency for the more complicated heterogeneous controller scenario with performance fluctuations. Through comprehensive performance evaluation, we demonstrate that our adaptive controller selection schemes can efficiently reduce response long-tail latencies and accommodate various system environments including the request-skews, the fluctuation of service rates and so on.

References

[1] T. Taleb, A. Ksentini, and R. Jantti, "'Anything as a service' for 5G mobile systems," *IEEE Netw.*, vol. 30, no. 6, pp. 84–91, Nov. 2016.

[2] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, "Mobile edge computing potential in making cities smarter," *IEEE Commun. Mag.*, vol. 55, no. 3, pp. 38–43, Mar. 2017.

[3] Y. Zhou, D. Zhang, and N. Xiong, "Post-cloud computing paradigms: A survey and comparison," *Tsinghua Sci. Technol.*, vol. 22, no. 6, pp. 714–732, 2017.

[4] M. Kablan, A. Alsudais, E. Keller, and F. Le, "Stateless network functions: Breaking the tight coupling of state and processing," in *Proc. 14th USENIX NSDI*, Boston, MA, USA, Mar. 2017, pp. 97–112.

[5] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz, "Service function chaining in next generation networks: State of the art and research challenges," *IEEE Commun. Mag.*, vol. 55, no. 2, pp. 216–223, Feb. 2017.

[6] P. Zave, R. A. Ferreira, X. K. Zou, M. Morimoto, and J. Rexford, "Dynamic service chaining with dysco," in *Proc. ACM SIGCOMM*, Los Angeles, CA, USA, 2017, pp. 57–70.

[7] D. L. C. Dutra, M. Bagaa, T. Taleb, and K. Samdanis, "Ensuring end-to-end QoS based on multi-paths routing using SDN technology," in *Proc. IEEE Global Commun. Conf.*, Dec. 2017, pp. 1–6.

[8] N. Gude *et al.*, "NOX: Towards an operating system for networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, 2008.

[9] A. Ksentini, M. Bagaa, T. Taleb, and I. Balasingham, "On using bargaining game for optimal placement of SDN controllers," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–6.

[10] J. Xie, D. Guo, Z. Hu, T. Qu, and P. Lv, "Control plane of software defined networks: A survey," *Comput. Commun.*, vol. 67, pp. 1–10, Aug. 2015.

[11] D. Kreutz, F. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[12] A. Ksentini, M. Bagaa, and T. Taleb, "On using SDN in 5G: The controller placement problem," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–6.

[13] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, 2008.

IEEE Proof

[14] P. Berde *et al.*, "ONOS: Towards an open, distributed SDN OS," in *Proc. ACM HotSDN*, 2014, pp. 1–6.
[15] A. Panda, W. Zheng, X. Hu, A. Krishnamurthy, and S. Shenker, "SCL: Simplifying distributed SDN control planes," in *Proc. 14th USENIX NSDI*, Mar. 2017, pp. 329–345.
[16] D. Hugheshallett, A. M. Gleason, and W. G. Mccallum, *Calculus: Single and Multivariable, Student Solutions Manual*, 6th ed. Hoboken, NJ, USA: Wiley, 2013.
[17] A. Bremler-Barr, Y. Harchol, and D. Hay, "OpenBox: A software-defined framework for developing, deploying, and managing network functions," in *Proc. ACM SIGCOMM*, Salvador, Brazil, Aug. 2016, pp. 511–524.
[18] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal, "Balanced allocations," *SIAM J. Comput.*, vol. 29, no. 1, pp. 180–200, 1999.
[19] M. Mitzenmacher, A. W. Richa, and R. Sitaraman, "The power of two random choices: A survey of techniques and results," in *Handbook of Randomized Computing*, vol. 11. Norwell, MA, USA: Kluwer, 2000, pp. 255–312.
[20] R. Nishtala *et al.*, "Scaling memcache at Facebook," in *Proc. USENIX NSDI*, 2013, pp. 385–398.
[21] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," in *Proc. ACM Conf. Emerg. Netw. Experim. Technol.*, 2013, pp. 283–294.
[22] T. Taleb and A. Ksentini, "Follow me cloud: Interworking federated clouds and distributed mobile networks," *IEEE Netw.*, vol. 27, no. 5, pp. 12–19, Sep./Oct. 2013.
[23] T. Taleb, "Toward carrier cloud: Potential, challenges, and solutions," *IEEE Wireless Commun.*, vol. 21, no. 3, pp. 80–91, Jun. 2014.
[24] T. Taleb, A. Ksentini, and A. Kobbane, "Lightweight mobile core networks for machine type communications," *IEEE Access*, vol. 2, pp. 1128–1137, 2014.
[25] J. Kempf, B. Johansson, S. Pettersson, H. Lüning, and T. Nilsson, "Moving the mobile evolved packet core to the cloud," in *Proc. IEEE 8th Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, Oct. 2012, pp. 784–791.
[26] T. Taleb *et al.*, "EASE: EPC as a service to ease mobile core network deployment over cloud," *IEEE Netw.*, vol. 29, no. 2, pp. 78–88, Mar./Apr. 2015.
[27] J. Xie, D. Guo, X. Zhu, B. Ren, and H. Chen, "Minimal fault-tolerant coverage of controllers in IaaS datacenters," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2017.2753260.
[28] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *Proc. USENIX OSDI*, 2010, pp. 351–364.
[29] A. Aissioui, A. Ksentini, A. M. Gueroui, and T. Taleb, "Toward elastic distributed SDN/NFV controller for 5G mobile cloud management systems," *IEEE Access*, vol. 3, pp. 2055–2064, 2015.
[30] M. F. Bari *et al.*, "Dynamic controller provisioning in software defined networks," in *Proc. Int. Conf. Netw. Service Manage.*, Oct. 2013, pp. 18–25.
[31] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," in *Proc. ACM HotSDN*, Hong Kong, Aug. 2013, pp. 7–12.
[32] Y. Zhou *et al.*, "A load balancing strategy of sdn controller based on distributed decision," in *Proc. IEEE 13th Int. Conf. Trust, Secur. Privacy Comput. Commun.*, Sep. 2014, pp. 851–856.
[33] Q. Mao and W. Shen, "A load balancing method based on SDN," in *Proc. 7th Int. Conf. Meas. Technol. Mechatronics Autom.*, Jun. 2015, pp. 18–21.
[34] D. Palma *et al.*, "The queuepusher: Enabling queue management in openflow," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, Sep. 2014, pp. 125–126.

Junjie Xie received the B.S. degree in computer science and technology from the Beijing Institute of Technology, Beijing, China, in 2013, and the M.S. degree in management science and engineering from the National University of Defense Technology, Changsha, China, in 2015, where he is currently pursuing the Ph.D. degree. He is also a joint Ph.D. student with the University of California, Santa Cruz (UCSC), CA, USA, from 2017. His study in the UCSC is supported by the China Scholarship Council. His research interests include distributed systems, data-center networks, and software-defined networks.

Deke Guo received the B.S. degree in industry engineering from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 2001, and the Ph.D. degree in management science and engineering from the National University of Defense Technology, Changsha, China, in 2008. He is currently a Professor with the College of Information System and Management, National University of Defense Technology, and a Professor with the School of Computer Science and Technology, Tianjin University. His research interests include distributed systems, software-defined networking, data center networking, wireless and mobile systems, and interconnection networks. He is a member of the ACM.

Xiaozhou Li received the Ph.D. degree in computer science from Princeton University in 2016. He is currently a Software Engineer with Barefoot Networks, where he is building new networking systems with fast programmable network chips. His research improves the performance, scalability, and efficiency of datacenter services, with a particular focus on combining new hardware and infrastructure capabilities with careful architectural design and algorithm engineering.

Yulong Shen received the B.S. and M.S. degrees in computer science and the Ph.D. degree in cryptography from Xidian University, Xian, China, in 2002, 2005, and 2008, respectively. He is currently a Professor with the School of Computer Science and Technology, Xidian University, and also an Associate Director of the Shaanxi Key Laboratory of Network and System Security. He has also served on the technical program committees of several international conferences, including the NANA, the ICEBE, the INCoS, the CIS, and the SOWN. His research interests include wireless network security and cloud computing security.

Xiaohong Jiang received the B.S., M.S., and Ph.D. degrees from Xidian University, China, in 1989, 1992, and 1999 respectively. From 2005 to 2010, he was an Associate professor, Tohoku University. He is currently a Full Professor with Future University Hakodate, Japan. His research interests include computer communications networks, mainly wireless networks and optical networks, network security, and routers/switches design. He has authored or co-authored over 300 technical papers at premium international journals and conferences, which include over 70 papers published in top IEEE journals and top IEEE conferences. He was a recipient of the Best Paper Award of IEEE HPCC 2014, IEEE WCNC 2012, IEEE WCNC 2008, IEEE ICC 2005-Optical Networking Symposium, and IEEE/IEICE HPSR 2002. He is a Member of IEICE.

IEEE Proof