Hindawi
Security and Communication Networks
Volume 2019, Article ID 1949343, 11 pages
https://doi.org/10.1155/2019/1949343

Research Article

An Active Controller Selection Scheme for Minimizing Packet-In Processing Latency in SDN

Haisheng Yu, Keqiu Li, and Heng Qi

School of Computer Science and Technology, Dalian University of Technology, No. 2, Linggong Road, Dalian 116023, China

Correspondence should be addressed to Heng Qi; hengqi@dlut.edu.cn

Received 24 March 2019; Accepted 19 August 2019; Published 13 October 2019

Academic Editor: Tom Chen

In software-defined network, the use of distributed controllers to control forwarding devices has been proposed to solve the issues of scalability and load balance. However, the forwarding devices are statically assigned to the controllers in these distributed systems, which can overload some controllers while others are underutilized. In this paper, we propose an architecture named ASLB (active controller selection load balance), which proactively selects appropriate controllers for load balancing and minimize packet processing delays. We also present a novel active controller selection algorithm (ACS) for ASLB that efficiently schedules traffic from the switch to the controller and designs an intermediate coordinator for actively selecting a controller to serve a request. We built a system and evaluated it on a physical platform. The results show that ASLB is much better than the static allocation scheme in terms of minimizing latency, bandwidth utilization, and throughput.

1. Introduction

Software-defined networking (SDN) revolutionizes the networking industry by enabling programmability, easier management, and faster innovation. Many of these advantages are achieved through their centralized control plane architecture. SDN enables realistic and controlled network management by decoupling the control and data planes [1–6]. To support large-scale networks, the control plane is implemented as a distributed system that must meet scalability, availability, and reliability requirements (e.g., ONOS [7], ONIX [8], OpenDaylight [9], HyperFlow [10], and ROSE [11]).

In these distributed systems, switches are statically assigned to one or more controllers. From a load balancing point of view, it is difficult to adapt to changes in traffic load on the control plane. Due to the static allocation between the switch and the controller, the controller will be overloaded if a large number of packets come from the switch assigned to the controller while the other controllers remain underutilized (more details will be found in Section 2). From a fault-tolerant point of view, it limits the ability of the network to react quickly. Networks (such as data center networks and enterprise networks) show significant changes of time and space requirements. It goes down when any delay or congestion on the network prevents the switch from sending messages to one or more controllers. When one controller fails in service, the other controllers need to select another one controller or the other controllers to manipulate the switches under the control of the failed controller.

This paper explores an active controller selection in which the fastest controller is selected to process the packet-in. Active controller selection focuses on the efficient handling of packets. When a packet does not match any rule, the switch will send the packet (called packet-in in the later article) to the controller. Later in this article, the request also refers to packet-in. The active controller selection is considered as an optimization to minimize the packet-in processing time. Later in this article, packet-in processing time also refers to response time. In active controller selection, each switch packet-in is sent to multiple controllers, so we propose active controller selection algorithm (ACS) to handle controller selection. When ACS chooses a controller to serve a packet-in, load balancing and runtime load sharing can achieve while minimizing the delay of the packet-in.

In order to verify the correctness of ACS, we design ASLB, a SDN network controller selection system. ASLB is an active controller selection architecture to reduce packet processing delay in software-defined networks. In the ASLB architecture, we designed an intermediate central coordinator that carries many functions between the switch and the controller, as shown in Figure 1. The most important responsibility is to schedule packets from the switch to the controller. With a coordinator, a switch will be served by multiple controllers at the same time; even if some controllers fail or in use, other controllers with better service capabilities can take it over. At the same time, the coordinator will choose the fastest and most reliable controller to serve each packet-in. As a result, ASLB improves reliability, scalability, and optimal resource utilization, even with some fault controllers.

In this paper, we first propose ASLB to process packet-in in a parallel distributed processing way. Then, we propose a novel controller selection algorithm (ACS), based on water-filling and use ranking scoring system to avoid the effects of herd behavior. To validate ASLB, we design and implement a prototype system, an intermediate coordinator that combines the controller selection algorithm with the implementation mechanism. Arranging the packet on the fastest and most reliable controller is the most challenging and important responsibility of the coordinator.

The paper is organized as follows. We describe the challenge of controller selection in Section 2. We present the design of active controller selection in Section 3. Then, we give the ASLB implementation in Section 4. Finally, we evaluate the performance of ASLB in Section 5 and conclude it in Section 6.

2. The Challenge of Controller Selection

In this section, we discuss the issue of passive controller selection and highlight the necessity and design-related challenges of active controller selection in an SDN environment.

2.1. Difficulties of Passive Controller Selection. To fulfill a large-scale network environment and achieve a scalable control layer, many recent papers have explored the architecture for building a large-scale or global-scale SDN controller [12–16].

Currently, the switch is assigned to a master controller and several slave controllers [17] and sends all packet-ins to the master. That means the switch will be obliged to obey the orders from one selected slave controller if the master stops working for overloading. So, load balancing of each controller is done passively. When a controller has overburden, coordination controllers will appear and choose a new master controller to balance the load.

HyperFlow [10] uses a logically centralized but physically distributed controller with switches connected to the physically closest parts of the controller, which update other controllers on the network via a publish/subscribe system. By passively synchronizing the entire network view of the controller, HyperFlow can formulate decisions to individual controllers to minimize control plane response times for data plane requests. However, if a HyperFlow controller fails, those switches controlled by the controller must be reconfigured to connect to a new controller. Reconfiguring a controller to a switch is a heavy event that can degrade the performance of the controller and switch, because the new controller needs to learn about the switch configuration and existed forwarding tables in the process.

ElastiCon [18] addresses controller failures by proposing a dynamic migration protocol between controllers and implements a dynamic load balancing system based on this protocol. However, there are some limitations to switch migration: the network will get worse due to the greater pressure placed on the control plane by switch migration. When a switch migration occurs, the main controller of the switch S changes from controller A to controller B, causing controller B to overload or crash. In addition, when S is connected to B, S does not work properly, which can also cause network congestion. Beehive simplifies the process of running a distributed controller, which transforms a centralized application into a distributed system, but Beehive still has load balancing problem when a particular module in the controller encounters a large number of requests [19].

Kandoo [20] uses a two-tier controller, a root controller, and a local controller, to reduce the load on the controller. Local controllers handle frequent events, while logically centralized root controllers handle rare events. Kandoo does reduce the load on the root controller but does not reduce the load on the local controller. The local controller still overloads or crashes because of the unbalanced packet-in load on the controller.

Although the proposals above have solve the problem of excessive load on some controllers, but they are built on increasing the load of other controllers that are easily overload or crash. So, how can we avoid the static allocation problem between the controllers and the switches when taking full advantage of the computational power of all controllers without overloading a single controller? In the next section, we will explain the reason we use active controller selection and how it works.

2.2. Motivation of Active Controller Selection. In recent years, SDN technology has been widely used to improve network efficiency; however, the issue of load balancing has not yet been solved in the research community. Solutions to load balancing can be generally classified into three categories: (1) some studies have been proposed to improve network performance by overprovisioning controllers at expected peak loads. Obviously, overprovisioning controllers can solve the load balancing by preparing enough backup controllers for each switch. However, that method has a high cost and energy consumption and packet arrival rate can reach almost 1-2 orders of magnitude [17, 21], which has basically been abandoned. (2) Some recent studies propose load balancing by migrating switches to the controller one by one. It balances the load on the controller by balancing the number of switches connected to the controller. We call this

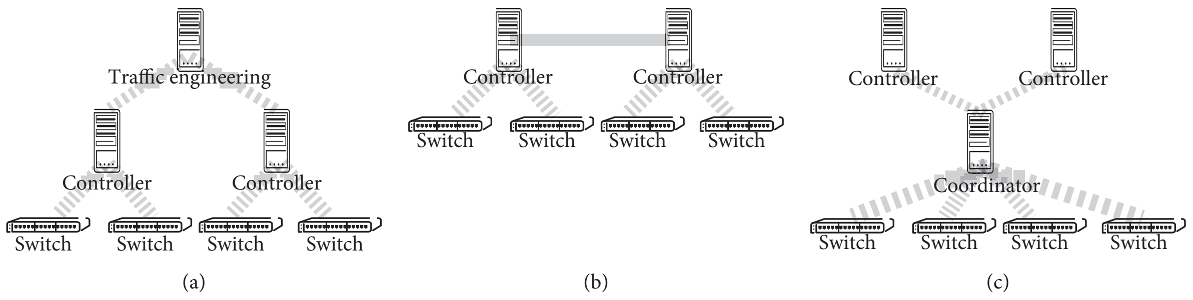

Figure 1: Three models to achieve load balancing: (a) CCLB achieves load balancing through centralized traffic engineering, (b) SFLB balances the load on the controller by transferring the load to other controllers, and (c) ASLB uses active controller selection to achieve load balancing.

method switch-shift-load-balance (SFLB). However, to some degree, it may be default due to unbalanced requests from the switches and will be further discussed in Section 2. (3) B4 [13] achieves load balancing through a centralized traffic engineering, which is called centralized-control-load-balance (CCLB). And, it balances the load on the controller by collecting all the traffic and then optimizing it. CCLB needs to collect information from each controller and then transfer it to a centralized computing module in order to calculate the optimization solution. The whole procedure takes longer response time than SFLB.

As we know, the processing power of a single controller is limited. OpenDaylight throughput with 16 threads is 215K, while ONOS average throughput with Goldeneye is 190K [22]. NOX [3] is capable of processing 30K requests (the request here is packet-in) per second, while Floodlight can handle about 250,000 requests a second. Maestro [23] has the strongest processing power and can handle about 300,000 requests per second.

However, if the load exceed the processing capacity of one controller, all types of controllers will fail. Consider the system exemplified in Figure 2, where Controller 1 and Controller 2 have the same processing capabilities of 300K requests. Assume each switch sends a burst of 150K requests to one of the two controllers. In Figure 2(a), the requests from three switches that connected to Controller 2 are 450K, which exceeds the processing capacity of Controller 2 (i.e., 300K). On the contrary, Controller 1 still has 150K requests processing capacity remaining, because it only receives 150K requests from Switch 1.

In order to solve the above load imbalance situation, ElastiCon proposes an elastic distributed controller architecture in which the controller pool is dynamically grown or shrunk according to traffic conditions and the load dynamically shift across controllers. As shown in Figure 2(b), 150K requests from switch 2 are shift from controller 1 to controller 2 with a switch migration protocol. After the switch migration, the system realize a load balance. This architecture has been implemented by many distributed SDN controllers (i.e., ONOS) [15, 24]. It gets load balance by assigning the same or nearly same number of switches to one controller. We call this method switch-shift-load-balance (SFLB), which means that it balances the load on the controller by balancing the number of switches connected to the controller.

However, SFLB will default fail due to unbalanced requests from the switches. As shown in Figure 2(a), both controllers are connected to two switches, but the load on controller 1 is still beyond what it can handle. Controller 1 needs to process 310K requests from Switch 1 and Switch 2 in total. On the contrary, controller 2 only has 170K requests to be processed from which another two switches are connected to. In this case, recent research proposes to load balance the network with centralized traffic engineering (i.e., B4 [13]). B4 is a private WAN connecting Google data centers across the planet. B4 has a centralized SDN gateway which consolidates topology events (i.e., the number of requests from the switch) from multiple controllers and a central traffic engineering (TE) server which calculate the optimization strategy. Figure 3(b) is a simple model of B4 in which controller 1 and controller 2 satisfy load balancing again through centralized traffic engineering. We call this method centralized-control-load-balance (CCLB), which means that it balances the load on the controller by collecting all the traffic carried by the controller and then optimizing it.

ASLB will certainly introduce additional latency and overhead, but the latency of ASLB is negligible compared to the latency of processing a packet-in. For example, Floodlight average responses/sec with 8 switches gives a figure of 18,001 [25], so the average latency of Floodlight is 55.5 ms. In ONOS, the latency for rerouting 1000 flows is 71.2 ms. The ONOS median throughput is 18,832 paths/sec when processing path installation [7]. Therefore, the ONOS median path installation latency is 53.1 ms. ASLB handles each packet-in with a delay of 0.25 ms or less, less than 0.47% of the latency of processing a packet-in. These data are calculated based on the minimum latency of 53.1 ms for a packet-in processing by the two controllers (each controller connect 8 switches). Compared to the latency of connecting 8 switches, when each controller connects more switches, the latency of the controller processing each packet-in will be further increased. It can get a big return at a small price which is the main reason we study on ASLB.

Compared with SFLB and CCLB, ASLB has the following advantages: (1) ASLB provides per-flow granularity of control, while SFLB can only be scheduled on a forwarding devices and CCLB only supports statistics-based control. (2) To achieve network load balancing, ASLB has less overhead

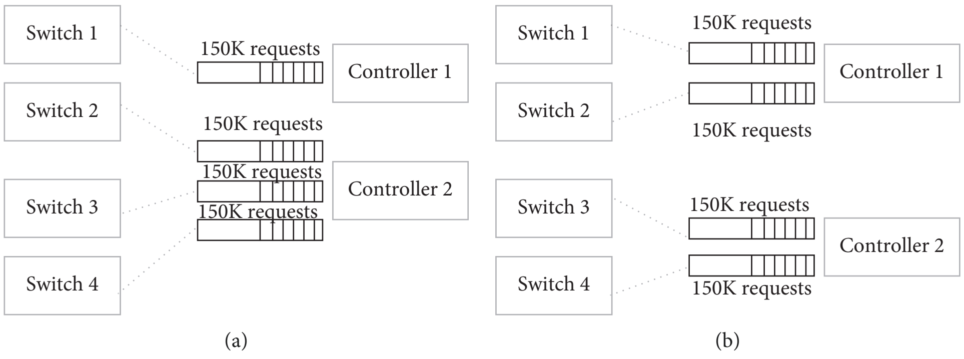

Figure 2: (a) Unbalanced switch collection cause congestion in controller 2. (b) With switch migration, the system realizes load balance again.

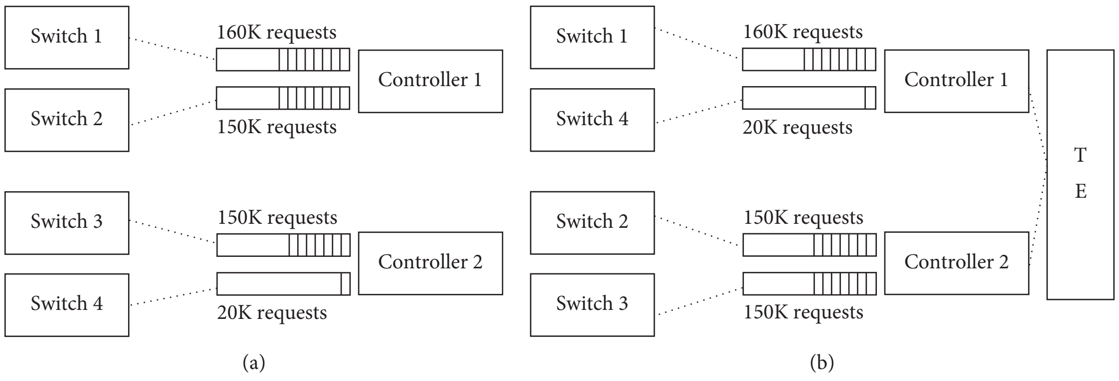

Figure 3: (a) Unbalanced requests cause congestion in controller 1. (b) With traffic engineering, the system realizes load balance again.

than CCLB and SFLB. SFLB needs to delay or terminate the work on the switches and the controllers during switch migration. Before making a decision, CCLB needs to collect the traffic of the whole network and then calculate to get the strategy that needs to be implemented. (3) ASLB load balancing works better because scheduling is based on flows, so the load that is assigned to each controller is more average and the resources are more utilized. (4) ASLB has faster response times compared to CCLB and SFLB. Because SFLB is a passive solution and it does not have a centralized control module, it needs to wait until the controller has a heavy load before it will start the switch migration. The risk of SFLB is that the switch migration process may not have completed and the controller crashed. The CCLB, on the other hand, needs to collect information from each controller and then hand it over to a centralized computing module to calculate the optimization solution (such as TE), which takes longer response time than SFLB and ASLB. In summary, ASLB focuses on the benefits of both SFLB and CCLB solutions, which deliver faster response times than SFLB, while gaining the benefits of centralized control. To sum up, ASLB has the advantages of both SFLB and CCLB, that is, it has the fastest response time and can obtain the optimization effect of centralized control.

Except for some special cases, all the above methods can partially solve the problem of load balancing in the control plane. In Figure 4(a), Controller 1 receives the 310K request from Switch 1, exceeding the controller 1's processing power. This article presents a coordinator to solve this problem. In Figure 4(b), a coordinator receives 110 packets from a switch and receives 30 packets from the other three switches. The coordinator can then send 70 packets to each controller and the controller will reach load balancing.

In this paper, our focus is on development of an active controller selection system to minimize packet-in processing latency, by allowing multiple controllers to process packet-ins from a lot of switches in a parallel way. Our approach builds on recent technology trends and also recognizes the need for incremental deployment to enhance the processing capacity of SDN controllers [12, 13, 26–30]. We believe that software-defined networking shows great promise for simplifying network management and enabling new networked services by implementing multiple controllers. Multiple controller existence enforces thousands of switches to send packet-ins to the most efficient controllers to ensure high quality of service in the network.

3. Milestone and Design of Active Controller Selection

An efficient active controller selection algorithm (ACS) should meet following requirements: meet switch time

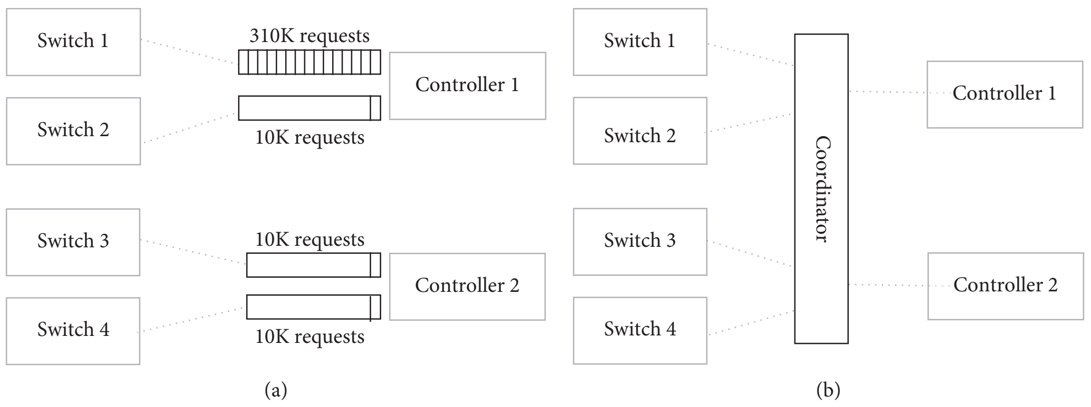

Figure 4: (a) Unbalanced packet-in queue cause congestion in controller 1. (b) With coordinator, the system realizes load balance again.

requirement, minimize packet-in latency, ensure fault tolerance, and adapt to controller fluctuation.

3.1. Milestone of Controller Selection. In a distributed SDN system, there may be multiple controllers handling traffic at the same time. So how to choose the right controller to improve system processing efficiency?

Here, we describe the five-step improvement of the controller selection algorithm.

Step 1. Random selection: random selection means that when a batch of requests arrives, the coordinator randomly selects a controller for a request. When the system has some switches and few controllers, the random selection is sometimes more efficient. When a random selection is made, the cost of acquiring controller load information is zero, so the controller selection cost is also lower.

Step 2. Power of two choices: random selection is very effective when the network is small but decreases when the network becomes large or the controller load is unbalanced. For example, when the network has two controllers A and B, the random selection algorithm chooses a full-load A controller to handle a request. Since the idle B controller is not used, the average controller efficiency decreases at this time. To prevent this from happening, we use the power of two choice algorithm [31] to solve this problem. Under the power of two choice algorithm, the coordinator first selects two or more controllers for a request and then distributes the requests to the controller with the lowest load.

Step 3. Batch-sampling: when using the power of two-choice algorithm, since each task needs to get the load information of the two or more controllers, this undoubtedly increases the cost of controller selection. So how to reduce this cost? Here, we use the batch-filling algorithm which helps reduce the overhead of getting the controller load. When a batch of m requests arrives, the coordinator does not process the requests one by one but instead gets the load information from $2m$ controllers and selects the m controller from the $2m$ controller to handle the m requests. The batch-sampling algorithm helps reduce the overhead of getting the controller load.

Step 4. Batch-filling: the batch-sampling algorithm first probes the load of the $2m$ controller and then selects the m lower-load controller to handle the m task, which means that each controller handles one task. But, sometimes this algorithm is inefficient. For example, m controllers have two controllers, A and B, with A load of 10% and B load of 90%. Here, we assume that handling each task will increase the controller by 10%. Obviously, if the tasks assigned to the B controller are passed to the A controller, the efficiency will be higher. It is like pouring water into m cups of different water levels. First, it is necessary to fill the cup with the lowest water level until the cup level is aligned with the penultimate cup level. Then continue the process. We call this algorithm batch-filling. Batch-filling algorithms can increase the system load balancing and make it more efficient.

Step 5. ACS: ACS is improved based on C3 [32] and water-filling [33]. Water-filling separates the response time from the queue length which may lead to herd behavior (we will further discuss it in Section 3.2). After combining C3 and water-filling algorithm, we can select the server with the fastest response time and no herd behavior.

3.2. Design of Controller Selection. We first define the notation in ASLB by the algorithmic model.

N identical controllers are supposed to be the integral part of system along with centralized coordinator. Each switch creates M packet-ins that require batch processing. The ACS assigns M packet-ins to the coordinator, and M packet-ins are processed into multiple batches. Batch size plays tricky role in the performance. In order to reduce the complexity, we maintain the same batch size in this paper although ACS has stable performance with variable batch sizes. ACS is based on water-filling [33] and uses ranking scoring system to avoid the effects of herd behavior [32]. Herd behavior leads to load oscillations, wherein multiple switches are coaxed to direct packet-ins towards the least-loaded controller and degrade the controller performance, which subsequently causes switches to repeat the same procedure with a different controller. Water-filling is the state-of-art traffic scheduling algorithm which acts like pouring water into a container in series according to water level, but it does not consider the effects of herd behavior.

According to this, at arrival of m packet-ins, the coordinator randomly probes dm controllers to acquire their queue lengths. The m packet-ins are dispatched to the dm controllers using water-filling technique, each packet-in is assigned sequentially to the lowest loaded controller, where the queuing sequence get updated after sending a new packet-in to a controller.

ACS is depicted in Algorithm 1 and operates as follows. V is a set of controllers and their corresponding ServerLoad. ServerLoad is a set which includes response time rt and packet-in queue length pq. The packet-ins will be sent to the servers which have the lowest ServerLoad by water-filling method. First, we should find the set K that includes all the lowest load servers. Then, we should find the set L that includes all the second lowest load servers. $Prod$ presents the packet-ins will be sent to servers. If $prod$ equals to m, all the packet-ins will be sent to the server. Otherwise, next cycle will begin because the servers in K is not enough to process m packet-ins. There may be several members in K that have the same server load. If K has more than one member, for example, g servers, the algorithm will divide sending list into g parts and every server in K gets one part from the list. The algorithm will find the distance between K and L and send packet-ins to K until the server load of K equals to L.

4. ASLB Implementation and Deployment

In this section, we present the ASLB implementation details and deployment. Figure 5 shows how ALSB works.

We implemented ASLB based on OpenVirteX [34], because the function of OpenVirteX is as an OpenFlow controller proxy between an operator network and SDN controllers. It means that an OpenVirteX node is both a switch and a controller (specifically, the PhysicalSwitch in OpenVirtex is the controller of the real physical switch; the VirtualSwitch in OpenVirtex is the switch of the controller). In OpenVirtex, a tenant's OS used to control a virtual network created by OpenVirtex. For ASLB, we modified the tenant module which controls the number of requests pushed to the buffer of each controller.

Given that in ASLB, there are as many controller groups as nodes themselves, and we need as many backpressure queues and controller selection schedulers as there are nodes. Thus, every read-request upon arrival in the system needs to be asynchronously routed to a scheduler corresponding to the request's controller group. Lastly, when a coordinator node performs a remote read, the controller handles the packet-ins and tracks the service time of the operation and the number of pending packet-ins in the controller. This information is piggybacked to the coordinator and serves as the feedback for the controller ranking.

There are challenges in this kind of efficient implementation. For one, how should we obtain the number of the working packet-ins in a SDN controller? When a packet-in accesses to ASLB, a slave coordinator will take charge of the packet-in to the storage of its packet-in queue buffer; meanwhile, the packet-in is given to a controller based on the algorithm. After the packet-in has been manipulated, the controller needs to remove the packet-in from the queue. A packet-in is marked primarily by DPID (data path id) and buffer ID, because buffer ID is a self-incremental number. Later, buffer ID is sent by pica8 switch, in which the buffer ID is also the same self-incremental number.

So, we have the packet-in queue length added one at a packet-in arriving time; then, we have the packet-in queue length reduced to one as having been manipulated to receive the packet-out. However, there remains a problem that if some packet-ins have no responses, they will resend the packet-ins, which will lead to increase packet-in queue length gradually. So, the queue should be emptied routinely.

For another, since a single remote peer can be a part of multiple controller sets, multiple admission control schedulers may potentially contend to push a request from their respective backpressure queues towards the same endpoint. Cautiousness should be considered to avoid starvation. To handle this complexity, every per-controller-group scheduler is represented as a single actor, and we configure the underlying Java thread dispatcher to fair schedule between the actors. The design of multiple backpressure queues also increases robustness, as one controller group enters backpressure, and will not affect other controller groups.

5. Evaluation

Extensive experiments are conducted to evaluate the performance of our proposed ASLB architecture model in this section. We carried out different experiments to evaluate the scalability, throughput, capacity, and latency of ASLB. The evaluation verifies the improved performance of ASLB and active controller selection algorithm.

To access the scalability of the ASLB, large-scale experiments are presented in this subsection. We built four clusters, including three clusters in CERNET [35] and one cluster in CloudLab [36], where CloudLab interoperates with existing test platforms (including GENI and Emulab) to take advantage of hardware from dozens of sites around the world.

Each cluster in CERNET has a PICA8 P-3297 openflow switch and eight Dell R720 (i5 2.70 GHz CPU and 32 GB RAM) servers. The PICA8 P-3297 openflow switch is an openflow-enabled 48-port physical switch. As shown in Figure 6, we use P-3297 switch instead of software switch because when a large number of switches are emulated, software switch does not scale well. The cluster in CloudLab has 15 virtual machines which include 7 controller nodes

```
REQUIRE: $V = <i; ServerLoad>$ //set of controllers and their corresponding server load, server load is a set which includes response time $rt$ and packet-in queue length $pq$
REQUIRE: client inputs:
$Req_m$//request list, $m$ is the size of the list
(1) $sendList \longleftarrow \varnothing$
(2) $m \longleftarrow \mathrm{sizeof}(V)$
(3) $selectedList \longleftarrow 2^* m$ randomly selected member from $V$
(4) sortedList $\longleftarrow$ sort $selectedList$ in decreasing order of ServerLoad
(5) while $(m > 0)$ do
(6)     $K \longleftarrow [first(sortedList)]$
(7)     $L \longleftarrow [second(sortedList)]$
(8)     $sortedList \longleftarrow sortedList\text{-}K$
(9)     $prod \longleftarrow (rt_l * pq_l - rt_k * pq_k)/rt_l$
(10)    $sendList \longleftarrow Req\ [0, (prod - 1)]$
(11)    $leftList \longleftarrow Req\ [prod, (m - 1)]$
(12)    $m \longleftarrow (m - prod)$
(13)    send($sendList$, $K$)
(14) end while
```

Algorithm 1: ACS active controller selection algorithm.

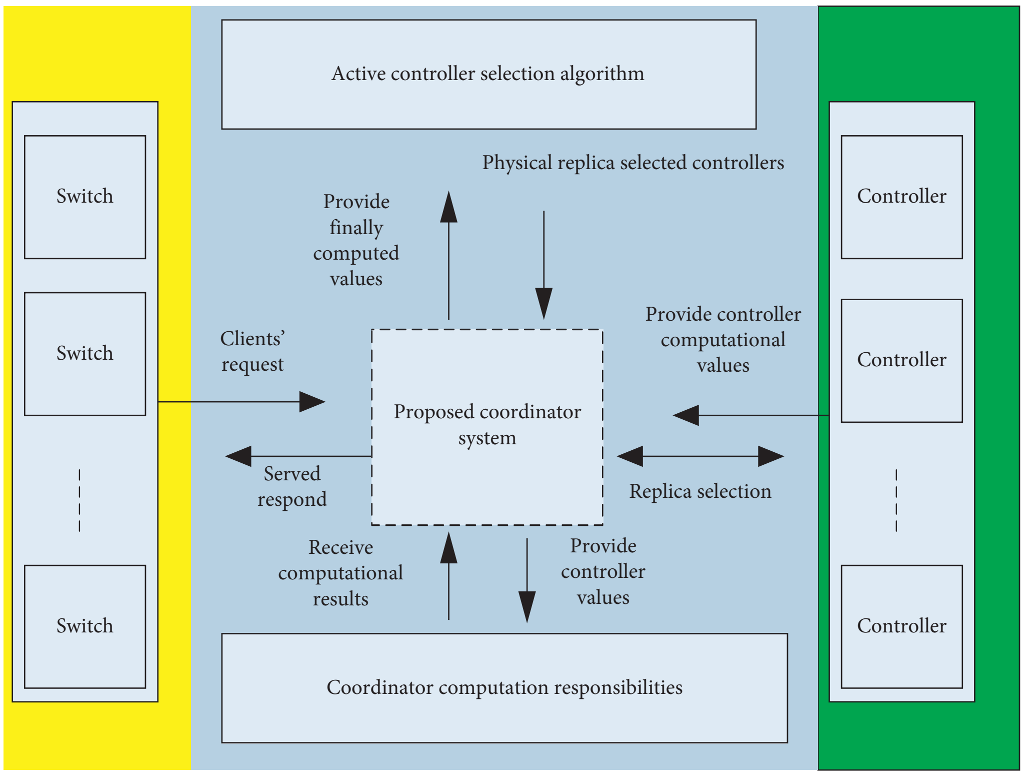
Active controller selection algorithm
Physical replica selected controllers
Provide finally computed values
Switch
Switch
Switch
Controller
Controller
Controller
Clients' request
Proposed coordinator system
Provide controller computational values
Served respond
Replica selection
Receive computational results
Provide controller values
Coordinator computation responsibilities

Figure 5: Proposed coordinator process of ASLB.

and 8 server nodes. One of the CERNET clusters is used to deploy ASLB instances and SDN controllers (named Control Cluster). Another CERNET cluster is used to emulate an SDN network (named Network Cluster). The other cluster is used for testing, and we deploy test tools in this cluster, such as iperf, cbench, and apache benchmark (named Test Cluster).

Three CERNET clusters are connected to each other via Ethernet, and CloudLab clusters connect to them over the Internet. To test the performance of an ASLB instance, we build an ASLB instance and 100 SDN controllers in Control Cluster and install the test tools in Test Cluster.

A number of metrics must be determined carefully to accurately reflect the ASLB performance. Thus, there are a

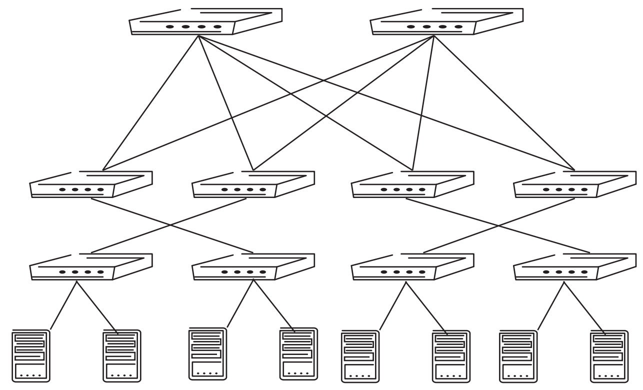

Figure 6: One cluster is a partial 4-radix FatTree. A PICA8 3297 switch is divided into ten switches. The number on the figure is the corresponding port number in the pica8 switch.

number of performance metrics that evaluate our ASLB performance, such as response time, bandwidth, throughput, and initializing time.

5.1. Response Time. The average of response time is computed to evaluate the overall system performance. The response time is related to the speed at which the controller handles packet-in. ASLB response time also refers to packet-in processing time and latency. To test the response time, we write an application which sends 100,000 flow table entry update requests and monitors the time used in processing the requests. More than 60% of the requests in ASLB is finished in 0.35 s, and more than 95% of the requests finished in 0.4 ms. In the worst case, a flow table entry update requests in ASLB can be finished in 0.5 ms. Figure 7 shows that thread modification percentage of the requests complete in a certain time. When there is only one thread, ASLB finishes 95% of the requests in 250 μs. When the thread number increases, ASLB uses a little more time to complete a request. As you can see from Figure 7, it wastes no more than 400 μs to complete 95% of the requests.

5.2. Bandwidth. Bandwidth reflects the network performance that means if the greater amount of data could be transmitted, the better the network performance is. In order to test the network bandwidth under the ASLB management, we conduct the bandwidth test by IPERF. We start a IPERF server on each host in Network Cluster, then start the running of IPERF clients on some hosts in Test Cluster. The bandwidth of the network is different when testing tcp and udp. Opening and closing the flow-mod also has a great impact on the network. Flow-mod is responsible for installing flow entries to the flow tables in SDN switches.

In order to test the impact of different response time on manipulating packet-in, we have made use of various kinds of controllers including Floodlight [2] and and ONOS [7]. Figure 8(a) illustrates the tested results of bandwidth in conducting UDP test by IPERF without opening flow-mod.

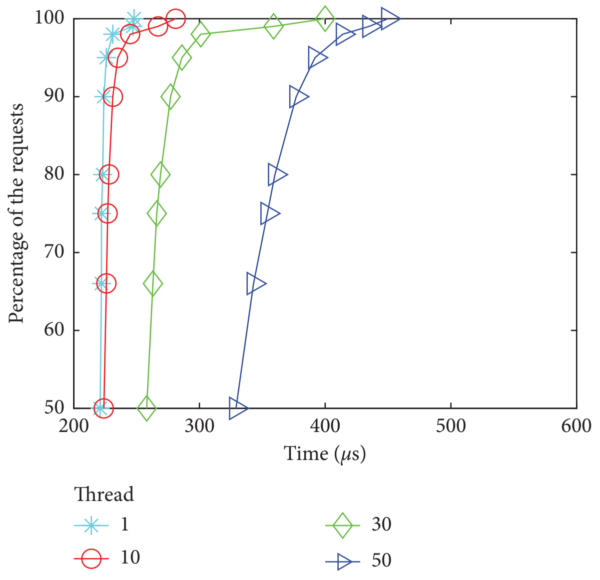

Figure 7: Thread modification percentage of the response complete in a certain time (μs).

To test the network bandwidth without flow-mod, the flow-mod is turned off and the controller will process all packets that do not match the flow table. As is shown in the Figure 8(a), the network bandwidth of ASLB is 4 times better than ONOS and Floodlight. Figure 8(b) shows the bandwidth in conducting UDP test by IPERF with flow-mod. With flow-mod, the network bandwidth managed by the CCLB, SFLB, and ASLB reaches to the peak value. Figure 9(a) shows the result of bandwidth in conducting the TCP test by IPERF without flow-mod. It is a condition to test the network bandwidth that all packet-ins will be manipulated by the controller if they do not matched with any flow entry in the flow tables. As is shown in the Figure 9(a), the network bandwidth of ASLB is slightly better than SFLB and CCLB respectively. CCLB performance is 20% less efficient than performance of SFLB and ASLB. Figure 9(b) provides the result of bandwidth in conducting TCP test by the means of IPERF with flow-mod.

5.3. Throughput. The throughput as regarded contains two aspects that the one is the coordinator throughput of ASLB and the other is the throughput of the network. In this paper, the throughput refers to the ASLB throughput, which reflects the processing capacity of the ASLB. We use cbench [37] to test the throughput and delay of coordinator. Figure 10 shows the throughput of the coordinator. The throughput of ASLB is 2.3 times as that of SFLB and 1.5 times as that of CCLB.

5.4. Initializing Time. Initializing time reflects the coordinator initializing time and the consumption time of link controller. By apache benchmark, we do pressure tests to evaluate the initializing time of the coordinator. We build a coordinator instance (flowN [38], FlowVisor [39], and OVX

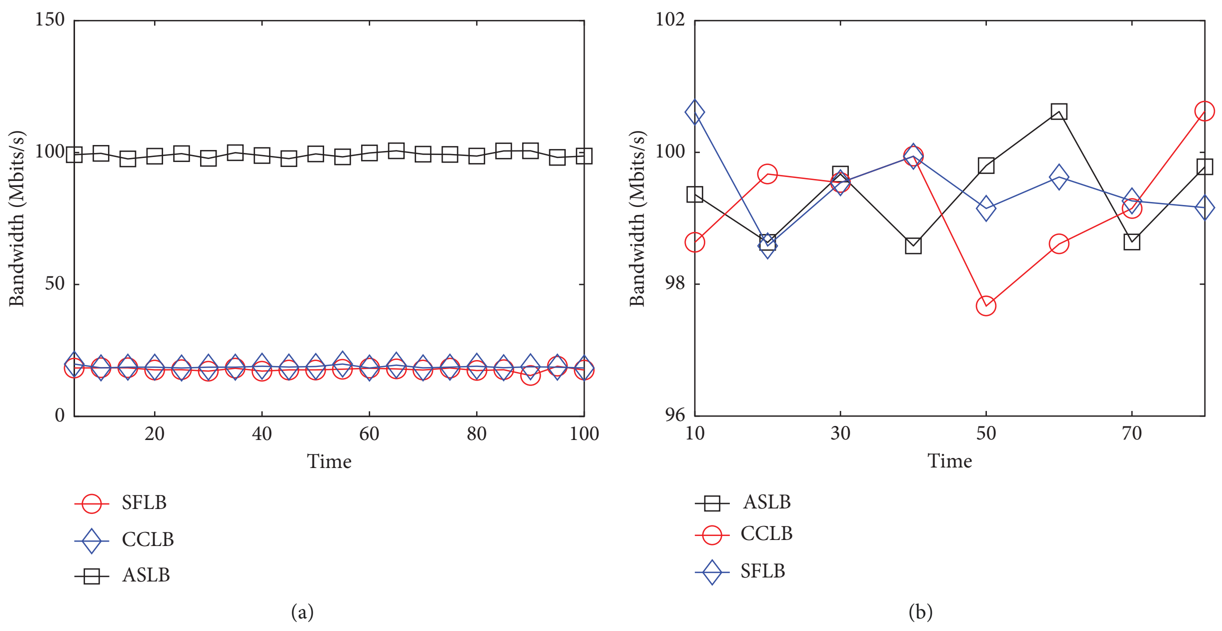

FIGURE 8: UDP bandwidth (a) without and (b) with flow-mod.

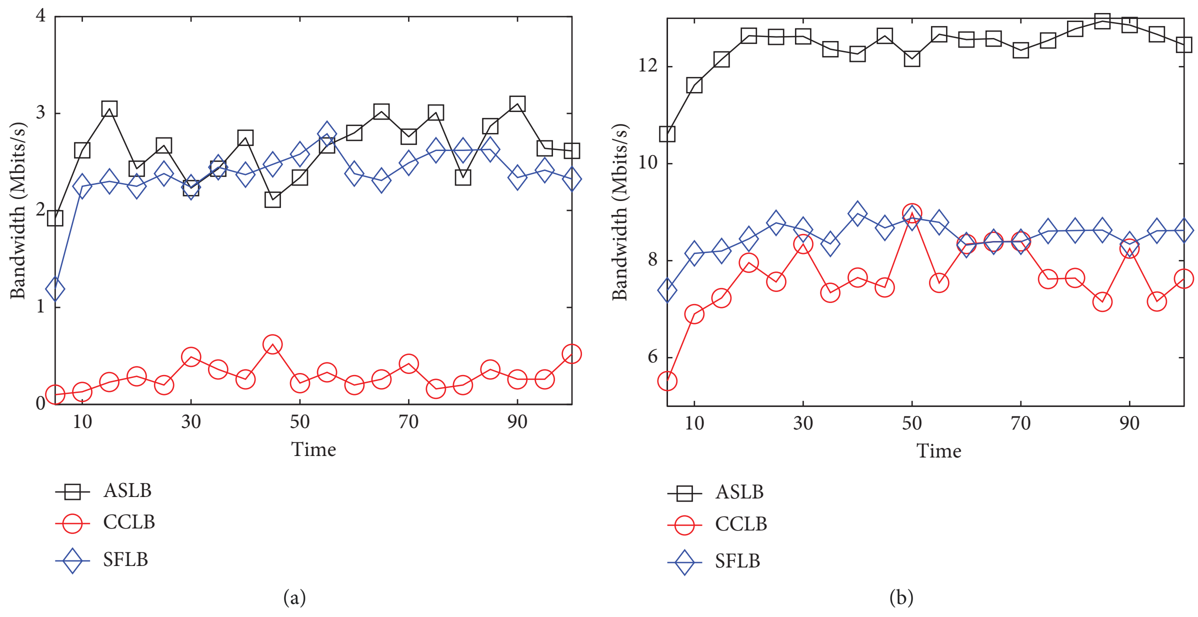

FIGURE 9: TCP bandwidth (a) without and (b) with flow-mod.

[34] instances) in Control Cluster and running apache benchmark in Test Cluster.

Figure 11 illustrates the controller initializing time of coordinator. The test result comes out of testcase 1, and the controller initializing time from which could be seen that if the number of threads is above 5, the consumption of it will be less than 6 ms, while the number of threads is below 5, the consumption of it will be less than 20 ms. Figure 12 indicates the latency of coordinator. The result of the graph comes out of testcase 1. As is shown in Figure 11, the processing delay of ASLB is lower than that of OVS, FlowVisor, and FlowN.

6. Conclusions

In this paper, we study on the optimization of controller selection that aims at minimizing the packet-in processing time. The main contributions made in this paper are as follows.

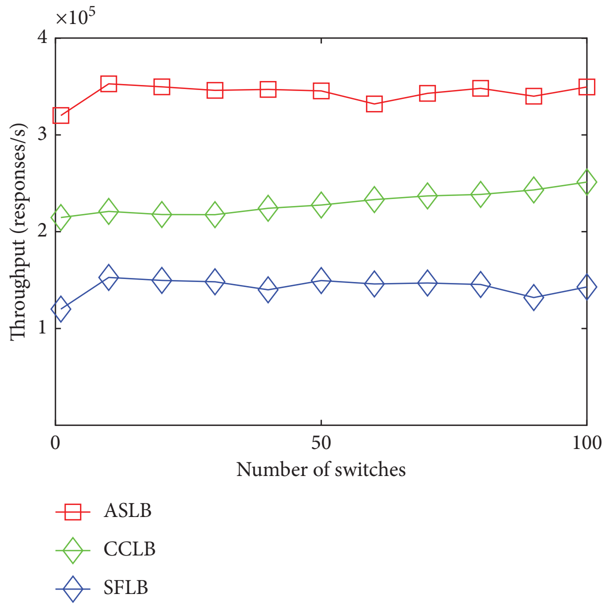

Figure 10: Throughput of ASLB, SFLB, and CCLB.

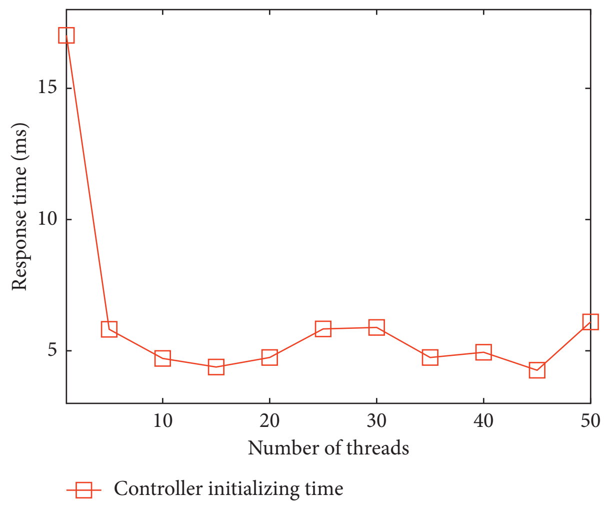

Figure 11: Controller initializing time of coordinator.

First of all, our research deals with packets from the switch in parallel distributed processing and addresses the problem of static allocation between the controller and the switch through active controller selection.

Second, we propose a state-of-the-art controller selection algorithm that selects the quickest and most reliable controller to handle the packet-ins from the switch.

Third, we design and implement ASLB and evaluate it on a real fat-tree platform.

On the basis of the designed ASLB architecture, we implement a coordinator to process packet-in by ACS and evaluate ASLB in a real testbed. The results of evaluation show that ASLB performs better than static assignment controllers in terms of latency, bandwidth utilization, and throughput. To further decrease the packet-in processing latency, we should enhance the ASLB architecture under the supervision of multiple coordinators and integrate coordinators into switches in the future research.

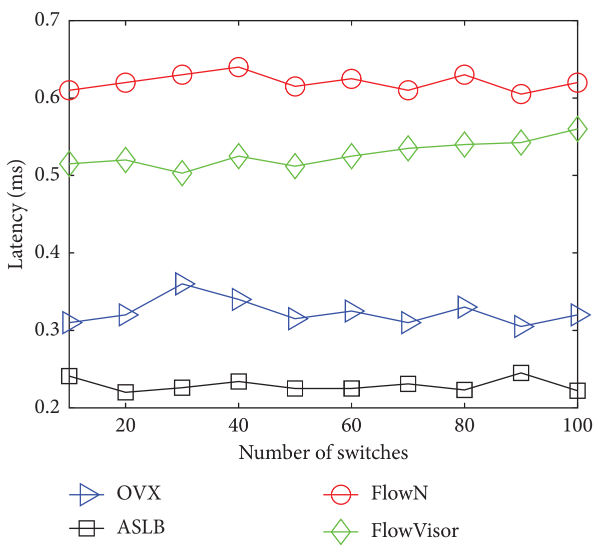

Figure 12: Latency of ASLB.

Data Availability

All the data is from my own expriments.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the State Key Program of National Natural Science of China (Grant No. 61432002), NSFC Grant Nos. 61772112, and the Science Innovation Foundation of Dalian under Grant 2019J12GX037.

References

[1] “Beacon project,” https://openflow.stanford.edu/display/Beacon/.
[2] “Floodlight project,” http://www.projectfloodlight.org/floodlight/.
[3] N. Gude, T. Koponen, J. Pettit et al., “NOX: towards an operating system for networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.
[4] “Pox project,” http://www.noxrepo.org/pox/about-pox/.
[5] “Ryu project,” http://osrg.github.io/ryu/.
[6] “Trema project,” http://trema.github.io/trema/.
[7] P. Berde, M. Gerola, J. Hart et al., “ONOS: towards an open, distributed SDN OS,” in *Proceedings of the ACM HotSDN*, Chicago, IL, USA, August 2014.
[8] T. Koponen, M. Casado, N. Gude et al., “Onix: a distributed control platform for large-scale production networks,” in *Proceedings of the USENIX Symposium on Operating Systems, Design and Implementation (OSDI)*, Vancouver, BC, Canada, October 2010.
[9] “Opendaylight project,” http://www.OpenDaylight.org/.
[10] A. Tootoonchian and Y. Ganjali, “HyperFlow: a distributed control plane for openflow,” in *Proceedings of the 2010*

Internet Network Management Conference on Research on Enterprise Networking, San Jose, CA, USA, 2010.

[11] T. Qiu, A. Zhao, F. Xia, W. Si, and D. O. Wu, "ROSE: robustness strategy for scale-free wireless sensor networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2944–2959, 2017.

[12] A. Greenberg, G. Hjalmtysson, D. A. Maltz et al., "A clean slate 4D approach to network control and management," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 3, pp. 41–54, 2005.

[13] S. Jain, A. Kumar, S. Mandal et al., "B4: experience with a globally-deployed software defined WAN," in *Proceedings of the ACM SIGCOMM*, Hong Kong, China, August 2013.

[14] T. Qiu, R. Qiao, M. Han, A. K. Sangaiah, and I. Lee, "A lifetime-enhanced data collecting scheme for the Internet of things," *IEEE Communications Magazine*, vol. 55, no. 11, pp. 132–137, 2017.

[15] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN control: survey, taxonomy, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 333–354, 2018.

[16] Y. Chang, A. Rezaei, B. Vamanan, J. Hasan, S. Rao, and T. N. Vijaykumar, "Hydra: leveraging functional slicing for efficient distributed sdn controllers," in *Proceedings of the International Conference on Communication Systems and Networks*, Bengaluru, India, 2017.

[17] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking-HotSDN'13*, Hong Kong, China, August 2013.

[18] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. R. Kompella, "Elasticon: an elastic distributed SDN controller," in *Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, Los Angeles, CA, USA, October 2014.

[19] H. F. Wedde, M. Farooq, and Y. Zhang, "Beehive: an efficient fault-tolerant routing algorithm inspired by honey bee behavior," *Ant Colony Optimization and Swarm Intelligence*, pp. 83–94, 2004.

[20] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proceedings of the ACM HotSDN*, Helsinki, Finland, August 2012.

[21] M. Yu, M. Yu, M. Yu, M. Yu, and M. Yu, "SilkRoad: making stateful layer-4 load balancing fast and cheap using switching asics," in *Proceedings of the Conference of ACM Special Interest Group on Data Communication*, pp. 15–28, Los Angeles, CA, USA, August 2017.

[22] M. Darianian, C. Williamson, and I. Haque, "Experimental evaluation of two openflow controllers," in *Proceedings of the IEEE ICNP*, Toronto, Canada, October 2017.

[23] Z. Cai, *Maestro: achieving scalability and coordination in centralized network control plane*, Ph.D. dissertation, Rice University, Houston, TX, USA, 2011.

[24] T. Hu, Z. Guo, P. Yi, T. Baker, and J. Lan, "Multi-controller based software-defined networking: a survey," *IEEE Access*, vol. 6, pp. 15980–15996, 2018.

[25] Z. K. Khattak, M. Awais, and A. Iqbal, "Performance evaluation of opendaylight SDN controller," in *Proceedings of the IEEE International Conference on Parallel and Distributed Systems*, pp. 671–676, Melbourne, Australia, 2015.

[26] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, "Software-defined internet architecture: decoupling architecture from infrastructure," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, Redmond, WA, USA, October 2012.

[27] A. Gupta, L. Vanbever, M. Shahbaz et al., "SDX: a software defined internet exchange," in *Proceedings of the ACM SIGCOMM*, Chicago, IL, USA, August 2014.

[28] T. Qiu, R. Qiao, and D. Wu, "EABS: An event-aware backpressure scheduling scheme for Emergency Internet of Things," *IEEE Transactions on Mobile Computing*, vol. 17, no. 1, pp. 72–84, 2018.

[29] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the one big switch abstraction in software-defined networks," in *Proceedings of the ACM Conference on Emerging Networking Experiments and Technologies*, Santa Barbara, CA, USA, 2013.

[30] Y. Sun, X. Hu, X. Liu, X. He, and K. Wang, "A software-defined green framework for hybrid ev-charging networks," *IEEE Communications Magazine*, vol. 55, no. 11, pp. 62–69, 2017.

[31] A. W. Richa, M. Mitzenmacher, and R. Sitaraman, "The power of two random choices: a survey of techniques and results," *Combinatorial Optimization*, vol. 9, pp. 255–304, 2001.

[32] L. Suresh, M. Canini, S. Schmid, and A. Feldmann, "C3: cutting tail latency in cloud data stores via adaptive replica selection," in *Proceedings of the USENIX NSDI*, Oakland, CA, USA, May 2015.

[33] L. Ying, R. Srikant, and X. Kang, "The power of slightly more than one sample in randomized load balancing," in *Proceedings of the 2015 IEEE Conference on Computer Communications (INFOCOM)*, Kowloon, Hong Kong, April 2015.

[34] A. Al-Shabibi, M. De Leenheer, M. Gerola et al., "OpenVirteX: make your virtual SDNs programmable," in *Proceedings of the ACM HotSDN*, Chicago, IL, USA, August 2014.

[35] "Cernet," http://www.edu.cn/.

[36] "Cloudlab," https://cloudlab.us/.

[37] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proceedings of the USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, San Jose, CA, USA, April 2012.

[38] D. A. Drutskoy, *Software-defined network virtualization with flown*, Ph.D. dissertation, Princeton University, Princeton, NJ, USA, 2012.

[39] R. Sherwood, G. Gibb, K.-K. Yap et al., "FlowVisor: a network virtualization layer," OpenFlow Switch Consortium, Technical Report, 2009.