Scalable and Crash-Tolerant Load Balancing Based on Switch Migration for Multiple OpenFlow Controllers

Chu Liang
Nagoya Institute of Technology
Nagoya, Aichi, 466-8555, Japan
liang@matlab.nitech.ac.jp

Ryota Kawashima
Nagoya Institute of Technology
Nagoya, Aichi, 466-8555, Japan
kawa1983@nitech.ac.jp

Hiroshi Matsuo
Nagoya Institute of Technology
Nagoya, Aichi, 466-8555, Japan
matsuo@nitech.ac.jp

***Abstract*—As the size of networks continues to increase, the scalability of the centralized controller becomes increasingly issues in Software Defined Networking. Distributed controllers have been proposed to solve the problem that the centralized controllers such as NOX and Floodlight suffer from. That logically centralized, but physically distributed architecture divide the network into zones with separate multiple controllers to achieve a better scalability control plane. However, Such distributed architecture introduces a new challenge to the load rebalancing of controllers when uneven load distribution among the controllers due to the statically configured mapping between a switch and a controller. Therefore, under variable traffic conditions in real networks, keeping load balance dynamically among the controller clusters is essential for realizing a high performance and scalability control plane. To address these issues, we propose a dynamic load rebalancing method based on switch migration mechanism for clustered controllers. The multiple controllers use JGroups to coordinate actions for switch migration. The whole network is divided into several groups and each group is set up one controller cluster. Our proposed method can dynamically shift the load across the multiple controllers through switch migration. The mechanism support controller failover without switch disconnection avoiding the single point of failure problem. We also implemented a prototype system based on OpenDaylight Hydrogen controller to evaluated the performance of our design. Our preliminary result shows that the method enables controllers to relieve the overload via switch migration and can improve throughput and reduce the response time of the control plane.**

***Keywords*-OpenFlow; Software-Defined Networking; Load Balance; OpenDaylight; JGroups;**

I. Introduction

Software Defined Networking (SDN) has emerged as a new paradigm that decouples the control plane from the data plane. SDN aims to offer easier management and faster innovation through dynamically customize the behaviors of a network by a centralized controller and programmable network devices.

In particular, OpenFlow [1], which standardized by the Open Networking Foundation (ONF), as one of the mostly representative protocol for SDN, carries the message between an SDN controller and the underlying network infrastructure. One of the most fundamental features of the OpenFlow protocol is the "packet-in" message. If a packet arrived at a OpenFlow-based switch does not match any forwarding rule, the packet can be configured as the "packet-in" message to be forwarded to the controller for corresponding policing processing [2].

However, with the growing number of OpenFlow-enabled devices, the massive traffic which from data forwarding plane to control plane can be outside the scope of the present specification, increases network load and makes the control plane a potential bottleneck. The traditional SDN implementation relies on a centralized controller (such as NOX [3] and Floodlight [4]) has several limitations related to performance and scalability. Therefore, some research work has proposed a logically centralized, but physically distributed control plane [5]–[7] across multiple controllers, which have better scalability and reliability, work cooperatively to partition the network into zones with separate controllers. These work try to address the global view and states consistency of distributed control plane. However, this multiple controllers approach introduces a new problem that a variation traffic arrive to clusters-based control plane leads to load imbalance among controllers which would result in suboptimal network performance. The traffic patterns are usually unpredictable in large data center due to elastic resources and flexible service. Prior work [8] has indicated that one cannot assume that applications are placed uniformly at random and shown that edge links in general show variation between peak and trough utilizations in data centers. Depending on where applications are generating flows or depending on the time of day, the traffic conditions can be significant variations in both temporal and spatial. If the switch to controller mapping conguration is static, some switches can generate a larger number of packet-in messages than other switches of the network. This condition implies the corresponding controller which these switches mapped to become overloaded, while other controllers remain under utilized.

To address this problem, in this paper, we propose a dynamic load rebalancing method for multiple OpenFlow controllers to improve the scalability of the control plane. According to the load conditions of controllers, our proposed method can dynamically shift the load across the multiple controllers through switch migration, which one controller changing the number of connected active switches by changing

978-1-4799-4152-0/14 $31.00 © 2014 IEEE
DOI 10.1109/CANDAR.2014.108

its role specified in OpenFlow 1.3 for a certain switch. The propose framework also support controller failover without switch disconnection avoiding the single point of failure problem. We also implemented a prototype system on top of the OpenDaylight Hydrogen controller [9] and the OpenFlow protocol 1.3 [2]. To evaluated its performance, we show an evaluation of control plane's response time and throughput, and compared with the traditional static mapping method. We also measure the performance cost of the switch migration and controller failover.

The rest of the paper is organized as follows: in Section II, related work is discussed. We present the proposed overview framework in Section III. The details of the implementation of proposed method is explained in section IV. We describe the evaluation setting and discuss the result of performance evaluation in Section V. We finally present our conclusions and future work in Section VI.

II. RELATED WORK

To address the performance challenge of control plane in SDN, multifaceted aspects should be taken into consideration:(1) maximize the performance of each physical controller machine. Such as Maestro [10], NOX-MT [11], Beacon [12] use this approach to achieve a high performance of control plane; (2) offload the controller by delegating some work to the forwarding devices. Devoflow [13], SDNC [14] actually reduce the overhead of the control plane by offloading the controller by delegating some work to the forwarding devices; (3) enable a cluster of controller nodes to achieve distributed control plane. Onix [5], Kandoo [7], and HyperFlow [6] use this approach to achieve a control plane with high scalability and reliability.

In more recent work [15]–[17] focusing on distributed controllers has also pursued the notion of elasticity in SDN controllers. ElastiCon [15] supports for elastic behavior which increases or decreases the number of controllers based on load estimates of control plane. [17] proposed a elastic approaches include dynamically changing the number of controllers and their locations under different conditions. [16] described an algorithm for dynamically adding or removing the controller in the cluster without any interruption to the network operation. [15], [16] also noted the need for dynamic assignment of switches to controllers and [15] proposed a mechanism to dynamically migrate switches among multiple controllers using the role-request message of OpenFlow 1.3, this research is complementary to ours, however, the seamless migration of a switch is based on a complex interaction between controllers. The detail of how to estimate the load of controller and wether support controller failover are also not described in the work. [16] also proposed a controller-switch mapping model with notation described, however, this model is not based on the role model of OpenFlow, and the controller failover based on IP alias makes the physical network routing configuration more complicated.

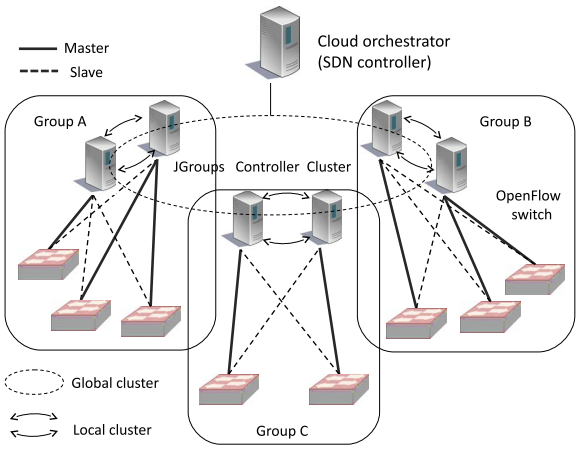

Fig. 1. Architecture of proposed method

III. ARCHITECTURE OF PROPOSED METHOD

The cluster based multiple OpenFlow controllers framework we proposed is shown in Figure 1. In this proposed framework, the whole network is divided into several groups and each group is set up one controller cluster, which called local cluster. All the local cluster make up a global cluster, which maintains the global view of the whole network.

The communication layer of our architecture is based on the middleware JGroups [18], which is used to provide flexible and reliable communication. JGroups based on multicasting is more efficient than unicasting in group communications, and the difference is magnified with an increasing number of controller nodes and is significant for scalability of the control plane. A further reason for using JGroups is that it provides group management functions, for example, the notification of change of cluster membership (joined/left/crashed node), etc. The features have allowed us to implement high level protocols (for example the interaction of switch migration and controller crash tolerance) with less overhead and therefore higher efficiency.

The multiple controllers in a local cluster coordinate each other by local JGroups channel to perform the switch migration, and the update of switch-controller mapping information stored locally by controllers is automatically synchronized amongst the global cluster via global JGroups channel. In our system, a coordinator node, which is determined primitively by JGroups, is elected as a special controller. The coordinator node is responsible for collecting load information and computing aggregate load of each controller node in the cluster. The switch-controller mapping information is stored locally by controllers and this mapping update is automatically synchronized amongst the global cluster via the global JGroups channel. Each controller node can be qualified for the role of coordinator and can be immediately replaced by one of the other nodes in the case of the coordinator node failure. Besides, if any node in the cluster is found to be inaccessible,

all other nodes can be aware of the change to update the switch-controller mapping.

The multiple controller functionality in OpenFlow 1.3 specifies three roles for a controller: master, slave and equal [2]. All the OpenFlow-based physical/virtual switches establish communication with multiple controllers to support controller failover and thus the switches are not required to re-establish connection in the event of controller outages. However, the switches only retain the role passively and the role-change mechanism in controller side is not provided in OpenFlow 1.3. Therefore, based on the multiple role functionality of OpenFlow 1.3, we proposed a method that different controllers can be set as master role for each individual switch to achieve load balancing for a multiple controllers cluster. When a switch connected to the controllers, the current lowest load controller will be master role for it to process packet-in messages from the switch, while others becomes slaves controller just keeping a connection. According to the load conditions of controllers, if load imbalance occurs, a switch from a heavily-loaded controller migrates to a lightly-loaded one. One controller can change the number of connected active switches by changing its role for a certain switch. By this switch migration method based on the proposed framework, the load of controller can dynamically shift across the multiple controllers to achieve desirable load balancing and improve the scalability of the control plane.

IV. DESIGN AND IMPLEMENTATION

In this section, we present the detail implementation of switch migration. We also show how the controller load is defined and estimated, and discuss the load-balancing model based on switch migration. Our management framework contains three modules and explained below:

1) Load Monitoring Module collects and calculates controllers load periodically and the coordinator maintains a global load info table.
2) Load Scheduling Module checks the global controllers load info table collected by the monitoring module and decides whether to perform a switch migration, which controller should be elected as master to receive load shifting and which switches should be selected to migrate.
3) Switch Migration Module in each controller coordinates actions for switch migration and changes switch-controller mapping.

A. Load Calculation

Here we consider two major types of load information for computing aggregate load of a server node based controller: switch input metric and server metric. The switch input metric includes the number of active switches connected to this controller Sn, the packet-in message requests rate $Sreq$, which come from all the active switches. The switch input load is calculated by the coordinator node. The server metric is all kinds of load information collected at servers, includes current CPU usage $Lcpu$, current memory usage $Lmem$, current network bandwidth usage $Lnet$. This part of load information is collected by each controller node and is provided to the coordinator node. We use the aggregate load value to indicate the real load information for controllers, but for different network condition, there can be different load information, therefore we can introduce coefficient R_i for different load information, to indicate the weight of different load information in aggregate load computing. The switch input load $Lswitch(C_i)$ can be calculated as follows:

$$L_{switch}(C_i) = R_1 \times \frac{\mathrm{Sn}(C_i)}{\sum_{i=0}^{n} \mathrm{Sn}(C_i)} + R_2 \times \frac{\mathrm{Sreq}(C_i)}{\sum_{i=0}^{n} \mathrm{Sreq}(C_i)},$$

The server load $Lserver(C_i)$ can be calculated as follows:

$$L_{server}(C_i) = [R_3\ R_4\ R_5] \begin{bmatrix} Lcpu(C_i) \\ Lmem(C_i) \\ Lnet(C_i) \end{bmatrix},$$

The aggregate load $LOAD(C_i)$ can be calculated as follows:

$$LOAD(C_i) = Lswitch(C_i) + Lserver(C_i),$$
$$\mathrm{i} = 0, 1, \cdots, n-1, \sum R = 1$$

While n expresses the number of controllers nodes in the local cluster and C_i expresses each controller. If the coefficient R_i cannot reflect the load of application well, system administrator can adjust them, until the right coefficients is found for current load condition.

The time interval to collect and calculate load information is determined by system administrator initially. The short time interval value can obtain an accurate load information while cause additional system overhead on server and communication overhead on the network at the same time. However, the long time interval value can not describe the load condition of controller accurately. Here, we use an adaptive load information collection algorithm. The adaption refers to the time interval to collect and calculate load information can be dynamically adjusted based on controller load changes, which can reduce system overhead, while obtain an accurate load information. This scheme is outlined in Algorithm 1.

$LOAD(C_i)$ and $LOAD^{'}(C_i)$ denote the aggregate load this time and last time respectively. In this paper, the load sampling interval timer is set to T_{max} initially, then the interval T is adjusted dynamically based on controller load changes.

B. Load Scheduling

Load Scheduling Module checks the global controllers load info table collected by the monitoring module and decides whether to perform a switch migration, which controller should be elected as master to receive load shifting and which switches should be selected to migrate.

1) Whether to perform a switch migration:

This decision depends on the load condition of controllers in a cluster. The difference load value of the heaviest-load controller and the lightest-load controller is defined as $Ldif_i$. When $Ldif_i$ is greater than a threshold value C, the switch migration is triggered. Note that the value C should be a

Algorithm 1 an adaptive load collection algorithm

$T \Leftarrow T_{max}$;
Set the sampling interval timer to T;
while $(true)$ **do**
  if (Timer trigger) $\|$ (Query initiatively) **then**
    Collect the local load information $Lserver(C_i)$;
    Send $Lserver(C_i)$ to coordinator;
    if is coordinator **then**
      Calculate the aggregate load $LOAD(C_i)$;
      Send $LOAD(C_i)$ to C_i;
      Update $LOAD(C_i)$ info table;
    end if
    $T = T_{\max} \Big/ (\Big|LOAD(C_i) - LOAD^{'}(C_i)\Big| + 1)$
    Set the sampling interval timer for T;
  end if
end while

appropriate value which can achieve a desired load-balancing level and avoiding frequent switch migration at the same time. In order to achieve a crash-tolerant control plane, the switch migration also can be triggered when a controller crash. The detail of this mechanism is explained in the following section.

2) which controller should be elected as master:
Before we describe the master controller selection, the coordinator node, a special role of controller is explained. The coordinator node is responsible for collecting load information of each controller node to computing aggregate load in the cluster. The coordinator node also maintains the view of switch-controller mapping and the load information table. The coordinator node is determined primitively by JGroups. Usually it is the first node attend in the cluster. Each controller node can be qualified for the role of coordinator and can be immediately replaced by one of the other nodes in case of the coordinator node failure. Different from the coordinator role, which usually not change, a controller is selected as the master controller by the coordinator node to receive load shifting by switch migration. Based on the load information table, the lightest-load controller is elected as the master controller for a certain switch who performs the migration. In order to support the scalability of control plane, the lightest-load controller in a cluster is also elected as the master when new switches are added to a network group.

Besides, when a controller crash, and if the controller owns at least one switch (e.g., S_i) as a master role, the lightest-load controller among the other controllers in the cluster is elected as the new master controller for S_i.

3) which switches should be selected to migrate:
Which switches should be selected to migrate has a important impact on load balancing. A switch whit a low packet-in request rate selected to migrate increases the time of switch migration to achieve a desired load balancing level. Contrary, if a switch whit a high packet-in request rate to be migrated, it can cause the overload to the new master controller. Here we proposed a dynamic round-trip time feedback based switch selection algorithm to decide the switch selection. When a switch is connected with controllers in cluster, the switch is assigned an initial weight W_i, which is an indicator of the switch selection priority. The weight is influenced by the round-trip time Rtt between the switch and master controller as follows:

$$W_i = W_i + A \times \sqrt[3]{RTT_i - RTT_i^{'}};$$

RTT_i and $RTT_i^{'}$ denote the RTT after and before switch migration respectively. A is an tunable coefficient. A bigger weight value denotes the switch is not the suitable selection in last switch migration, it should be selected with a high priority to migrate in the next switch migration.

The whole Load Scheduling algorithm is outlined in Algorithm 2.

Algorithm 2 Load Scheduling algorithm

while $(LOAD(C_i)Table \neq NULL)$ **do**
  SortByLoad$\{C_0, \cdots, C_n\}$
  if $|LOAD(C_0) - LOAD(C_n)| > C$ **then**
    SortByW_i $\{S_0, \cdots, S_n\}$ $(S_i \in C_i)$ as Master
    Migrate S_0 from C_0 to C_n;
  end if
  Update S_iC_i mapping
end while

C. Switch Migration

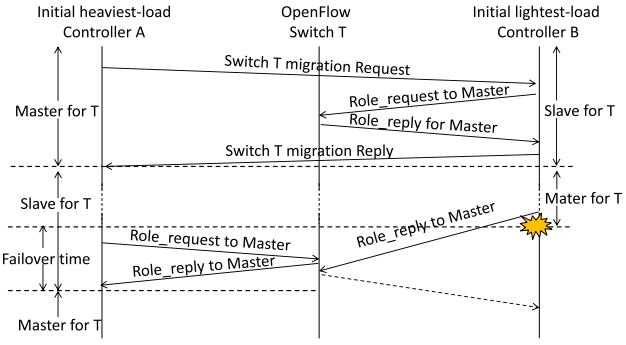

Fig. 2. Control procedure for switch migration

OpenFlow 1.3 specifies three roles for a controller: master, slave and equal [2]. By default, the slave controller does not receive switch asynchronous messages (e.g., packet-in). Each controller can send an `OFPT_ROLE_REQUEST` message to communicate its role to the switch, and change role at any time. When a controller changes its role to `OFPCR_ROLE_MASTER` for certain switch, the switch changes previous master controller to have the role `OFPCR_ROLE_SLAVE`. Based on the functionality, we proposed a method that different controllers can be set as master role for each individual switch to achieve load balancing for a multiple controllers cluster. Figure 2 shows an operational sequence of our proposed scheme. In a normal operation of

switch migration, the heavily-loaded controller A first sends a switch migration request message to a lightly-loaded controller B, denotes that it wants to migrate switch T to controller B. Then, controller B sends an OFPT_ROLE_REQUEST message to switch T for changing its role to MASTER. To respond to the OFPT_ROLE_REQUEST message, switch T replies the OFPT_ROLE_REPLY message, and starts to send asynchronous messages (e.g., packet-in) to controller B after the completion of the role-change process. Upon receiving the OFPT_ROLE_REPLY message from switch T, controller B replies the switch migration reply message back to controller A and updates the local switch-controller mapping Finally, controller A also updates the switch-controller mapping synchronically and the whole switch migration process is completion. Note that when the switch performs a role change, no reply message is generated to the controllers except the controller which sends a OFPT_ROLE_REQUEST message. Therefore, the mapping information is stored locally by controllers and this local update is automatically synchronized amongst the global cluster via the global JGroups channel.

If a controller (e.g., Controller B) is assumed to have failed, the JGroups channel detects the failure of controller and automatically notices the change to all the controllers. If the controller B owns at least one switch (e.g., switch T) as a master role, the lightest-load controller (e.g., Controller A) among the other controllers in the cluster sends an OFPT_ROLE_REQUEST message on it's own initiative to switch T according to mapping information for taking over the master role for switch T.

Note that if Controller B crashed as soon as it had sent OFPT_ROLE_REQUEST message to switch T, Controller A also sent OFPT_ROLE_REQUEST to message switch T according to the above mechanism almost at the same time. The two OFPT_ROLE_REQUEST messages can be out-of-order due to network delay and this condition can lead to a wrong master role transition. For example, the OFPT_ROLE_REQUEST message sent by Controller B firstly arrived at switch T later than the request from Controller A, switch T will regards Controller B as master, in fact that Controller B had crashed. Therefore, as a part of the master election mechanism, we assign a monotonically increasing counter with the OFPT_ROLE_REQUEST message when each new master is designated to detect out-of-order messages during a role-change transition.

V. EVALUATION

In this section, we evaluate the performance of the proposed method compared with a switch-controller static mapping model. Our experimental environment is illustrated in Figure 3. We set up two physical OpenFlow controller nodes (i.e., OFC A and OFC B) in one cluster. Note that we are primarily concerned with the control plane traffic load and need not emulate the high overhead data plane or actually transmitting packets through the data plane. Therefore, we chose Mininet [19], which emulates a network of software-based virtual OpenFlow switch as our experimental testbed. We set up a

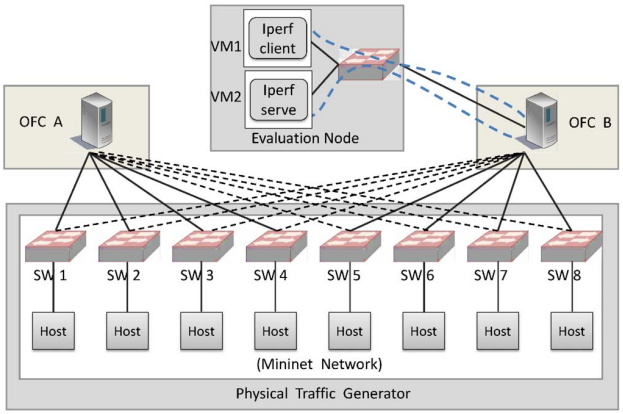

Fig. 3. Experimental environment

TABLE I
MACHINE SPECIFICATIONS

	Controller Node	Traffic Generator	Evaluation Node
OS	Ubuntu 12.04(3.5.0)		CentOS 6.5(64bit)
CPU	Core i5(4 core)	Core i7(1 core)	Core i5(4 core)
Memory	16GB	8 GB	
OpenFlow Switch	-	Open vSwtich-1.10.0	
Network	100Mbps Ethernet		

total of eight edge OpenFlow switches connects to the both of two controllers. Each switch connect an end-host, which used as the traffic generator to initiate UDP flows to any other host in the network. In order to emulate the maximum flow arrival rate of control plane, we use the reactive approach which no flow forward rules are pre-installed in switches. In the control plane, we use the modified sample-forward application that disables the function of that installing the flow rule on switches. That means all the packet received at switches are treated as new flow, and transmitted to the controller as packet-in messages. We use a link separated physical node to evaluate the performance of our method. The measuring packets also go through an OpenFlow switch and are sent to the controller. The specifications in our testbed are illustrated in Table I.

We simulate three different workloads to stress controllers through adjusting the different sending rate of flows of each end-host by hping [20] tool. Before evaluating the performance of our method, we use Cbench [21] tool to measure the rate in which flow requests are handled by the controller. Our measurements show an average rate of 15328 packets per second (pps) can be handled by a single default controller. From this value, we design three kinds of workloads to evaluate our system, as illustrated in Table II.

In order to simulate the uneven load distribution among the controllers, in the beginning of each workload, we configured

TABLE II
THREE KINDS OF WORKLOADS

	switch 1 $\sim$ 2	switch 3 $\sim$ 4	switch 5 $\sim$ 6	switch 7 $\sim$ 8
Workload A	1000 pps	2000 pps	2000 pps	4000 pps
Workload B	1000 pps	2000 pps	4000 pps	6000 pps
Workload C	1000 pps	2000 pps	6000 pps	8000 pps

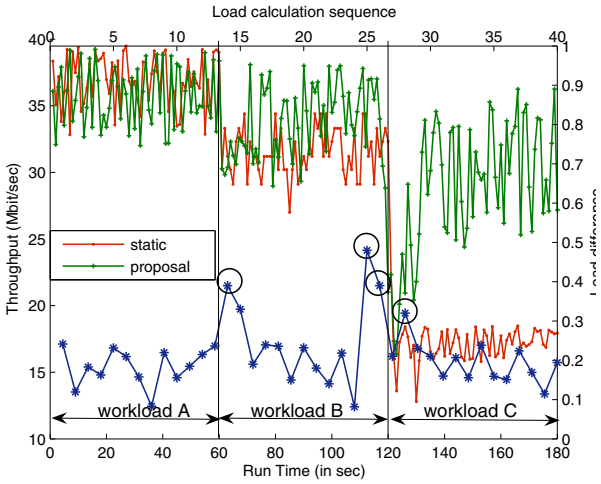

Fig. 4. Average maximum throughput

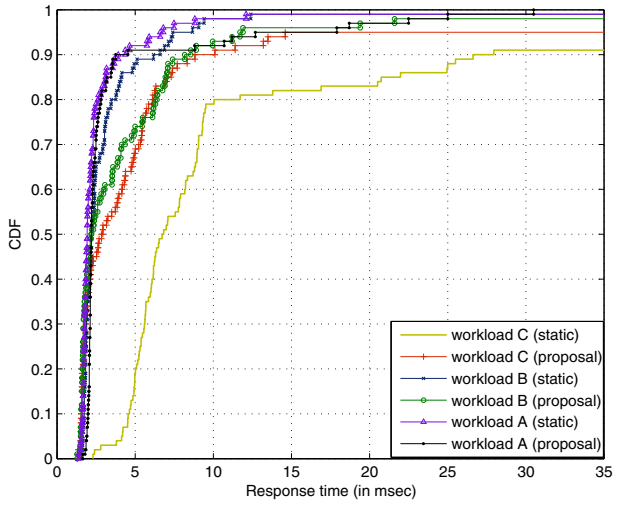

Fig. 5. Response time

switches 1 $\sim$ 4 to connect OFC A as master and OFC B as slave, meanwhile, switches 5 $\sim$ 8 to connect OFC A as salve and OFC B as master. This switch-controller mapping keeps static in each workload in order to compared with our proposed method.

A. Throughput

We used iperf [22] to evaluate and plot the mean throughput of the OFC A and OFC B with varying workloads as illustrated in Figure 4. From the result, we observe that in static mapping model, the mean throughput of controllers decreases under workload B and sharp decreases under workload C. That because the variance of packet request lead to skew of controller utilization, i.e. load imbalance among OFC A and OFC B. Under workload B, OFC B had heavy load and began to put the packet request into waiting queue. Under workload C, OFC B overloaded and the packets begin loss due to buffer overflow of OFC B. We also plotted the difference value of two controllers. In proposed method, as the figure shows, when the difference value of load is greater than a threshold value C (here is 0.3), the switch migration is triggered. OFC B dynamically shifted the heavy load to OFC A according switch migration, throughput of OFC A remains almost unchanged since the load imposed on it is still below its processing capability. The result shows, when OFC A and OFC B are equally loaded by switch migration, the average throughput settles stably.

B. Response Time

In a OpenFlow network where flow entry setup is performed reactively, the controller response time directly affects the flow completion times. We evaluate the response time of OFC B by hping command. Figure 5 plotted the response time CDFs of OFC B fixing the load level with workload A,B,C, and still compared with the static master/slave mapping model. As the figure shows, the workload significantly affected response time. The response time increases marginally up under workload B and goes up higher under workload C. That is because once the packet interval rate exceeded the capacity of the controller, queuing causes response time to shoot up.

C. Failover Time

Finally, we measure the cost of switch migration process. We observe the migration process takes about 2ms under workload A. The failover process showed in Figure 2 takes about an average of 20ms, which mostly affected by the failure detection based on heartbeat messages provided by JGroups. These show that the switch migration can be done quickly and even though the controller failover.

VI. CONCLUSION AND FUTURE WORK

The load imbalance among controllers results in suboptimal network performance. To address this problem, we propose a scalable and crash-tolerant load balancing based on switch migration for multiple OpenFlow controllers. According to the load conditions of controllers, our proposed method enables the controllers coordinate actions by JGroups to dynamically shift the load across the multiple controllers through switch migration. The switch migration mechanism can be done quickly and support controller failover. The result of evaluation showed that compared with the traditional static mapping, our method can improve the resource utilization of controller cluster, and improve the throughput and response time of control plane. We are continuing the optimization of the load scheduling modules, with focus on developing load scheduling algorithms that independent of any threshold value or conditioning parameter which determined by network administrator. In addition, we plan to implement a topology-aware switch migration algorithms which could improve the scalability in the real large scale network and to evaluate the performance in a variety of different application on the controller and topologies with more practical traffics.

REFERENCES

[1] "The open networking foundation," https://www.opennetworking.org.

[2] "Openflow switch specification 1.3.4," https://www.opennetworking.org/sdn-resources/onf-specifications.

[3] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[4] "Floodlight project," http://www.projectfloodlight.org.

[5] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks." in *OSDI*, vol. 10, 2010, pp. 1–6.

[6] 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*, 2010, pp. 3–3.

[7] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proceedings of the first workshop on Hot topics in software defined networks*, 2012, pp. 19–24.

[8] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010, pp. 267–280.

[9] "Opendaylight," http://www.opendaylight.org.

[10] A. L. C. Zheng Cai and T. S. E. Ng, "Maestro: A system for scalable openflow control," 2010.

[11] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, vol. 54, 2012.

[12] D. Erickson, "The beacon openflow controller," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013, pp. 13–18.

[13] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, 2011, pp. 254–265.

[14] J. C. Mogul and P. Congdon, "Hey, you darned counters!: get off my asic!" in *Proceedings of the first workshop on Hot topics in software defined networks*, 2012, pp. 25–30.

[15] 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*, 2013, pp. 7–12.

[16] V. Yazici, M. O. Sunay, and A. O. Ercan, "Controlling a software-defined network via distributed controllers," *arXiv preprint arXiv:1401.7651*, 2014.

[17] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks." in *CNSM*, 2013, pp. 18–25.

[18] "Jgroups project," http://www.jgroups.org.

[19] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, p. 19.

[20] "hping," http://www.hping.org.

[21] R. SHERWOOD and Y. KOK-KIONG, "Cbench: an open-flow controller benchmarker," http://www.openflow.org/wk/index.php/Oflops.

[22] "Iperf," http://iperf.sourceforge.net.