Redundancy Method for Highly Available OpenFlow Controller

Keisuke Kuroki, Masaki Fukushima, and Michiaki Hayashi
Integrated Core Network Control and Management Laboratory
KDDI R&D Laboratories, Inc.
Saitama, Japan
e-mail:{ke-kuroki, fukusima, mc-hayashi}@kddilabs.jp

Nobutaka Matsumoto
Department of Evolved Packet Core Network Development
KDDI Corporation
Tokyo, Japan
e-mail:nb-matsumoto@kddi.com

***Abstract*—OpenFlow is an important element for achieving Software Defined Networking (SDN) and is expected to be an enabler that solves the problems of today's network. Thanks to the centralized management with OpenFlow, agile network operation can be achieved with flexible programmability; however, the centralized management implies a significant impact of any outages of the OpenFlow controller. Hence, a high availability technology is indispensable for building the OpenFlow controller. To achieve the highly available system, we have to consider extraordinary events (e.g., power outage) affecting the entire data center as well as anticipated server failures within a local system. In this paper, we review the issue in using the conventional redundancy method for OpenFlow controllers. Based on this observation, we propose a redundancy method considering both local and global (i.e., inter data-center) recoveries using the multiple-controllers capability that is defined in OpenFlow switch specification version 1.2 and later. The proposed redundancy scheme eliminates virtual IP address-based redundancy and frontend server causing limitation of performance scalability, while it achieves competitive role change and failover times.**

Keywords-OpenFlow; controller; redundancy.

I. INTRODUCTION

This paper is an extended version of our previous work [1]. Towards future telecom services, the programmability of the network is expected to shorten the service delivery time and to enhance the flexibility of service deployment meeting diversified and complex user requirements on various applications (e.g., real-time and non real-time applications). Software Defined Networking (SDN) is an important concept for achieving a programmable network and OpenFlow [2] is an important factor for achieving the concept. OpenFlow is an enabler of the centralized management solution, which enables management and control of several OpenFlow switches, which allows the network operators to configure the switches easily and speedily. However, we have to solve some issues of OpenFlow (i.e., scalability, reliability and so forth) to deploy the OpenFlow technique in carrier grade networks. Many researches have addressed the issues of the OpenFlow-based solution.

Fernandez evaluates several OpenFlow controllers from the viewpoint of scalability in centralized management and control [3]. Message processing performances of two operation modes (i.e., proactive and reactive) of the OpenFlow controller are evaluated using several existent implementations (e.g., Floodlight, NOX, Trema). Pries et al. analyze the scalability of the OpenFlow solution for a data center environment to show an implementation guideline [4]. The paper concludes that, to achieve lossless and low delay performance in the data center application, the number of OpenFlow switches managed by one controller should be limited to eight. To leverage the advantage of centralized management, the OpenFlow controller should not be a simple flow switching policy server. OpenQoS [5] architecture delivers end-to-end quality of service (QoS) with OpenFlow-based traffic control. The OpenFlow controller with OpenQoS plays the role of collecting the network state to perform dynamic QoS routing, i.e., the controller has a route calculation function just like the Path Computation Element (PCE). Indeed, in the Internet Engineering Task Force (IETF), PCE architecture is growing as a stateful operation supporting the enforcement of path provisioning in addition to its original path computation role. Hence, the importance of the OpenFlow controller is growing with the broader concept of SDN, and thus the high availability of the controller system must be discussed.

There are two approaches to achieve high availability of the OpenFlow controllers. One approach is to reduce their load. The OpenFlow controller exchanges many messages with the OpenFlow switches especially in reactive mode. As a result, the OpenFlow controller could be overloaded and thus become unable to process incoming messages. In such a case, some processing is required to handle failover. If the OpenFlow Controller uses Link-Layer Discovery Protocol (LLDP) [6] messages to discover link and node failures and manages and monitors several switches, the monitoring model has serious scalability limitations. Kempf et al. [7] propose a monitoring function for OpenFlow switches that achieves a fast recovery in a scalable manner. Dixit et al. [8] propose a new OpenFlow switch migration algorithm for enabling load shifting among the OpenFlow Controllers. This algorithm improves the response time for the Packet-in messages by shifting the controlled switch. Thus, there are some researches on reducing the load of the OpenFlow controller for protection of the data-plane.

The other approach is to replace a single controller with redundant controllers. However, there is little research on the redundancy of the OpenFlow controller, which must play an important role in SDN.

In this paper, we investigate the issue of achieving redundancy for the OpenFlow controller with a conventional method, and we propose a method to improve the availabil-

ity of the OpenFlow controllers. In the proposed redundant method, "global" recovery (i.e., inter data-center redundancy) as well as local recovery (i.e., redundancy within a local network) are considered. The proposal achieves a competitive failover time compared with existing redundant schemes (e.g., server clustering), while the proposal does not require any frontend server limiting performance scalability of the OpenFlow controller.

The organization of this paper is as follows: In Section II, we review related works and the capability of multiple-controllers as defined in OpenFlow switch specification 1.2 [9] and also explain its applicability to achieving redundancy of the OpenFlow controller. In Section III, we describe a conventional method to achieve the redundancy of the OpenFlow controller by using the Virtual Router Redundancy Protocol (VRRP) and its limitation. In Sections IV-A and B, we propose the redundancy method using multiple-controllers in a single domain and evaluate its performance. In Sections IV-C and D, we propose the redundancy method using multiple-controllers in multiple domains and evaluate its performance. Finally, concluding remarks are given in Section V.

II. Background and Related Work

Typical implementation of OpenFlow allocates a controller separating the control plane from the data plane, and an OpenFlow switch playing the role of data plane communicates with an OpenFlow controller using the OpenFlow protocol over a Transport Layer Security (TLS) [10] or a Transmission Control Protocol (TCP) connection [11] defined as an "OpenFlow channel." The switch tries to forward a packet by looking up flow entries populated in advance by the controller. If the packet does not match the current flow entries, the switch sends a packet-in message over the OpenFlow channel to the controller in order to retrieve a direction on how to treat the packet.

One method of handling data plane failure is to implement a monitoring function on the OpenFlow switch; however, only the monitoring function in a data plane is not sufficient for achieving high availability in an OpenFlow network. We cannot achieve a highly available OpenFlow network without achieving the redundancy of the OpenFlow controller. In the case of controller outages, the OpenFlow channel is lost accordingly, and then the controller cannot successfully process the packet-in message. Hence, new packets that are not matched with the flow entry are simply dropped or allowed to fall in a default operation (e.g., forwarding to a neighbor anyway) that does not provide desirable services until the ultimate recovery of the controller. To achieve a high availability in the OpenFlow network, we have to achieve recovery methods in both global and local networks that exploit the redundancy of the OpenFlow controllers.

The HyperFlow [12] approach improves the performance of the OpenFlow control plane and achieves redundancy of the controllers. HyperFlow introduces a distributed inter-controller synchronization protocol forming a distributed file system. HyperFlow is implemented as a NOX-C++ application and synchronizes all events between controllers by messaging advertisements. In the case of controller failures, HyperFlow requires overwriting of the controller registry in all relevant switches or simply forming hot-standby using servers in the vicinity of the failed controller. Thus, this approach assumes re-establishment of the OpenFlow channel, and does not assume the multiple-controllers capability defined in OpenFlow 1.2. Therefore, the time duration of the failover operation may increase with the growth of the number of switches managed by the failed controller. Since the failover process of HyperFlow does not consider any server resource, overload of CPU utilization is a potential risk in the event of migrating switches to a new controller especially in the global recovery scenario.

There are several methods of general server redundancy, and such methods may also be effective for OpenFlow controllers. For example, one possible server redundancy can use one virtual IP address aggregating hot-standby or several servers. Koch and Hansen [13] evaluate a failover time in the case of using the virtual IP address-based implementation with the Common Address Redundancy Protocol (CARP), which is similar to VRRP [14]. According to the analysis, the average time to change the role between master and backup is 15.7 milliseconds. However, the virtual IP address-based approach may take a longer failover time in the case of applying this approach on the OpenFlow network because this approach involves the re-establishment process of the OpenFlow channels. We discuss this issue in Sections III and Sections IV-A. Although the virtual IP-based scheme is straightforward if it is applied within single LAN, it cannot simply be applied to multiple locations (e.g., data centers) managed under different addressing schemes. This means that the virtual IP-based scheme alone is not sufficient to tackle global recovery. Zhang et al. [15] propose a server clustering method with a mechanism to seamlessly handover the TCP connection between backend servers. While each TCP connection is visible to only one backend server in a normal clustering scheme, the proposal [15] makes the connection visible to at least two back-ends using proprietary backup TCP (BTCP) protocol within a backend network. The connection migrates to a backup, and then the backup is able to resume the connection transparently before the client TCP connection is lost. Using this scheme, the connections are recovered by the backup server within 0.9 seconds including a failure detecting time of 0.5 seconds. This approach is expected to be applicable also for global recovery involving multiple locations. However, from the viewpoint of the performance scalability of the OpenFlow controller as analyzed in [3, 4], a common frontend server required in the clustering system can be a serious bottleneck of message processing in the control plane (e.g., if the frontend server is broken, all TCP connections are lost). The high availability scheme should avoid such single frontend server to ensure the performance scalability of OpenFlow controllers. In addition, when we tackle global recovery with many switches, the migration process should also consider the server utilization. However, conventional approaches do not consider utilization of the server resources (e.g., CPU).

OpenFlow specification 1.2 introduced the capability of multiple-controllers by defining three states (i.e., MASTER, SLAVE, and EQUAL) of a controller. A controller plays its own role by using the multiple-controllers capability, and the state itself is owned by the switch. In the three states, MASTER and EQUAL have full access to the switch and can receive all asynchronous messages (e.g., packet-in) from the switch. A switch can make OpenFlow channel connections to multiple EQUAL controllers, but the switch is allowed to access only one MASTER controller. In the SLAVE state, a controller has read-only access to switches and cannot receive asynchronous messages apart from a port-status message from the switches. A controller can change its own state by sending an OFPT_ROLE_REQUEST message to switches. On receipt of the message, the switch sends back an OFPT_ROLE_REPLY message to the controller. If the switch receives a message indicating the controller's intent to change its state to MASTER, all the other controllers' states owned by the switch are changed to SLAVE. This function enables a switch to have multiple OpenFlow channels, and thus the switch is not required to re-establish new OpenFlow channels in the event of controller outages. In the multiple-controllers capability, the role-change mechanism is entirely driven by the controllers, while the switches act passively only to retain the role. Therefore, it is important to investigate the implementation of the controller side to achieve the redundancy; however, that has yet to be proposed. We use the capability of multiple-controllers to achieve high availability of the control plane.

III. CONVENTIONAL METHOD

In this section, we describe a conventional method of a redundant OpenFlow controller (OFC) using VRRP. We investigate the issue in the case of using VRRP. Table I shows the parameters common to all experiments (i.e., Section III, Section IV-A, and Section IV-C) in this paper, and Table II shows the parameters specific to the VRRP experiment in Section III.

We implement OpenFlow-1.2-compliant controllers and switches on Linux by extending an existing implementation [16], which consists of a NOX-based controller [17] and Ericsson TrafficLab 1.1 software switch [18]. In addition, we use Keepalived [19] to run VRRP between the controllers.

We conducted an experiment on our testbed as shown in Fig. 1. There are two controllers (i.e., OFC01 and OFC02). To achieve redundancy between the two controllers, VRRP is used. Initially, the state of OFC01 is set to *Master* and thus OFC01 has a virtual IP address. The state of OFC02 is set to *Backup*. An OpenFlow Switch (OFS01) is connected to OFC01 through an OpenFlow channel since OFC01 has a virtual IP address. OFS01 sends a packet-in message to the controller when it receives a new packet undefined in the flow entry because OFS01 is operated under the reactive mode. A traffic generator sends packets at the rate of 100 packets per second (pps).

Fig. 2 shows an operational sequence that indicates the state transition in the case of OFC01's going down. Initially,

TABLE I. PARAMETERS COMMON TO ALL EXPERIMENTS.

Node	Parameter	Value
OpenFlow controller	operating system	Ubuntu12.04
	openflow implementation	NOX-based [16]
	network interface	Gigabit Ethernet
OpenFlow switch	operating system	Ubuntu12.04
	openflow implementation	TrafficLab 1.1 software-based [16]
	network interface	Gigabit Ethernet
Traffic generator	sending rate	100 packets/s

TABLE II. PARAMETERS SPECIFIC TO VRRP EXPERIMENT.

Node	Parameter	Value
OpenFlow controller	vrrp implementation	Keepalived [19]
	vrrp advertisement interval	1000 ms
	vrrp master down interval	3004 ms

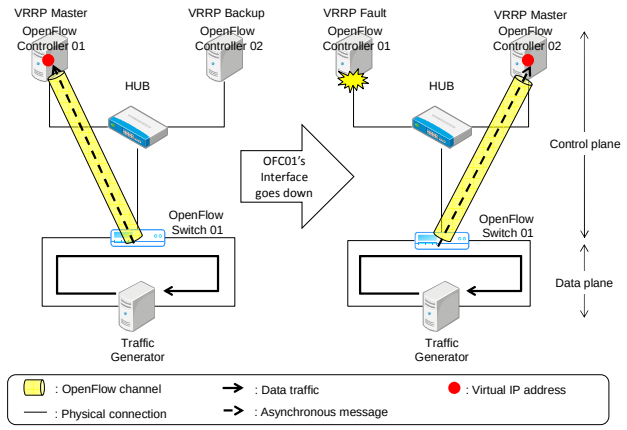


Figure 1. Experimental scenario using VRRP.

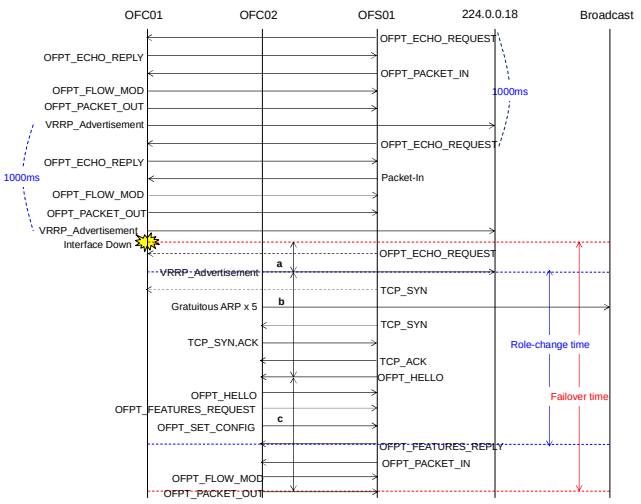


Figure 2. Operational sequence of the recovery using VRRP

the OFS01 sends an asynchronous message to OFC01 through the OpenFlow channel. Since the OFCs are running VRRP, OFC01 sends a VRRP advertisement message to OFC02 every 1000 ms. When OFC01 goes down, OFC02 sends a VRRP advertisement message to take over the virtu-

al IP address and change its own state to Master from Back-up after the master down interval, which is a guard timer for Backup to judge the failed condition of Master. Master down interval is defined by

3 * Advertisement_Interval + ((256 – Priority) / 256).

The master down interval for OFC02 is 3004 ms because the priority of OFC02 is set to 255 (i.e., the highest priority) to detect the failure of OFC01 as soon as possible.

OFC02 sends five gratuitous ARP packets to inform that the MAC address of the virtual IP address is changed in one second after the Keepalived sends the VRRP advertisement message. If OFS01 sends a SYN packet to reconnect to the virtual IP address before OFC02's sending gratuitous ARP packets, OFS01 cannot connect to OFC02 because the destination MAC address of the SYN packet is set to the OFC01's MAC address. If the OFS is successfully reconnected to OFC02, in other words, if TCP connection is reestablished, OFS01 starts sending a Hello message to OFC02 to establish an OpenFlow protocol connection. Then, OFS01 sends a packet-in message to and receives a packet-out message from OFC02. Thus, the failover is completed. In Fig. 2, the *failover time* is defined as the duration time from the failure event of OFC01 to the first packet-out message sent by OFC02. Also, the *role-change time* is defined as the duration time from OFS02's sending the VRRP advertisement message to the receipt of OFPT_FEATURE_REPLY by the OFS. Intervals *a*, *b* and *c* shown in Fig. 2 are defined as follows. Interval *a* is *advertisement delay* that is the time from OFC01's going down to OFC02's sending the advertisement message. Interval *b* is *TCP-recovery delay* that is the time from OFC02's sending an advertisement message to OFS01's sending the OPFT_HELLO message. Interval *c* is *OpenFlow-recovery delay* that is the time from OFS01's sending the successful OPFT_HELLO message to OFC02's sending OFPT_PACKET_OUT message.

We measured the failover time and role-change time 10 times respectively. The results are shown in Table III. We

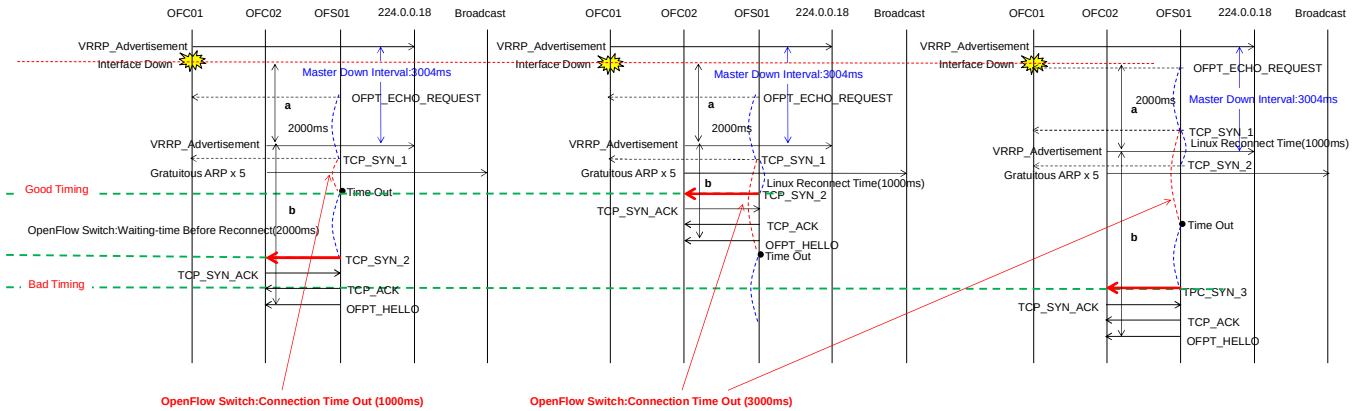
TABLE III. ROLE-CHANGE TIME AND FAILOVER TIME IN THE CASE OF DEFAULT PARAMETER.

	Minimum [ms]	Average [ms]	Maximum [ms]
Role-change time	3058	3365	3613
Failover time	5307	5653	5958

can improve these times by tuning some parameters.

Fig. 3 shows three operational sequence patterns of VRRP and Fig.3-(a) shows the sequence in the case of the default parameter. In VRRP, it is difficult to shorten the time to detect a failure because the minimum value of the master down interval is 3004 ms. To shorten the failover time in VRRP, we should reconnect the OFS to the OFC as soon as possible. To this end, OFS01 should send a SYN packet as soon as OFC02 sends the gratuitous ARP packets. The OFS01 first sends the SYN packet in two seconds after the failure of sending the OFPT_ECHO_REQUEST message. Since OFC01 is down, the OFS cannot receive the SYN_ACK packet. In our OFS implementation, the channel-establishment timer of OpenFlow is expired if both of the TCP connection and OpenFlow connection are not established within one second. And then OFS01 retries to connect to OFC02 after two seconds.

We changed the channel-establishment timer of OpenFlow to three seconds from one second. In that case, the SYN packet is retransmitted in one second after OFS01's sending the first SYN packet because the initial value of the TCP retransmission timer of Linux is one second. So, we can shorten the failover time as shown in Fig. 3-(b). However, the failover time depends on the timing of the failure of OFC01. If OFS01 sends an OFPT_REQUEST message to OFC immediately after OFC01 fails, the second SYN packet is sent before OFC02's sending the gratuitous ARP packets. As a result, the failover time increases as shown in Fig. 3-(c). Table IV shows the result in the case of changing the connection-establish timer to three seconds. According to Table IV, we can shorten the minimum and average times by changing the channel-establishment timer of OpenFlow.



(a) Channel-establishment timer is set to 1000ms

(b) Channel-establishment timer is set to 3000ms in good timing.

(c) Channel-establishment timer is set to 3000ms in bad timing.

Figure 3. VRRP-based switchover operations for three conditions of the channel-establishment timer.

TABLE IV. ROLE-CHANGE TIME AND FAILOVER TIME IN THE CASE OF CHANGING THE CHANNEL-ESTABLISHMENT TIMER.

	Minimum [ms]	Average [ms]	Maximum [ms]
Role-change time	1040	2621	4597
Failover time	3663	5161	7336

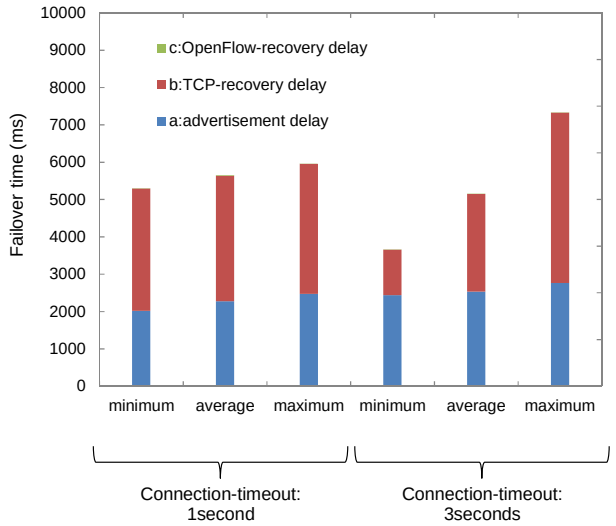


Figure 4. Breakdown of failover time.

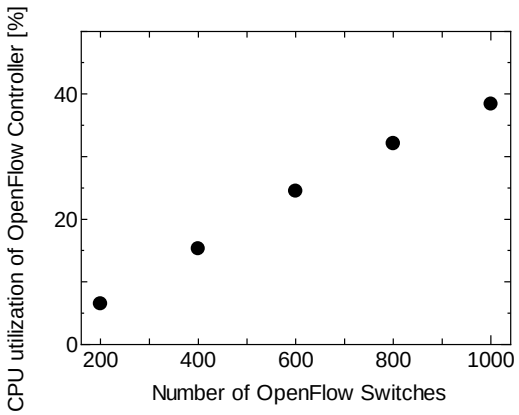


Figure 5. CPU utilization of NOX controller process during VRRP-based failover operation.

However, the maximum time is longer than the case with the original timer of one second.

Fig. 4 shows a breakdown of the failover time in both cases (i.e., the channel-establishment timer of OpenFlow is one second or three seconds). Intervals *a*, *b* and *c* set in Fig. 4 correspond to the markers shown in Fig. 2. Since Interval *c* is a very small value compared with Intervals *a* and *b*, the value is hardly visible in Fig. 4. According to Fig. 4, we can shorten Interval *b* by changing the channel-establishment timer of OpenFlow in the minimum and average value. However, considering that Interval *b* varies depending on the timing of OFC01's failure, it is difficult to adjust the parameter to the optimal values. In addition, it is also costly for network operators to set the optimal value to each switch if the OFC controls many OFSes provided by various vendors on various operating systems.

In the redundancy method using VRRP for the OFC, we measure the CPU utilization of the NOX controller process when the Master controller is changed and the OFS is reconnected to the controller. The experimental testbed is almost the same as shown in Fig. 1, except for the following two points. First, we use Open vSwitch [20] as the OFS to connect several switches to the OFC. Second, the traffic generator does not generate the data packet to measure only CPU utilization due to the failover of the NOX process. We measure the CPU utilization by a *top* command of Linux at one-second intervals. The maximum CPU utilization of the NOX process due to the failover is evaluated as a function of the number of OFSes. Fig. 5 shows the average of 10 measurements. According to Fig. 5, the CPU utilization of the NOX process increases with the growth of the number of OFSes. The CPU utilization is approximately 40% with 1000 OFSes.

In summary, using VRRP for redundancy of OFCs has two issues. First, it requires a long failover time. The failover time of VRRP has lower bound depending on its implementation. For example, Keepalived needs at least three seconds as the failover time since the minimum advertisement delay is two seconds and minimum TCP-recovery delay is one second. Also, it is difficult to shorten the failover time by changing parameters. Second, considering that the CPU utilization due to the failover process is high, VRRP is not suitable for a large OpenFlow network.

IV. PROPOSAL AND EVALUATION

In this section, we propose an architecture that uses multiple-controllers capability for local and global recoveries. We also evaluate recovery operation in two scenarios (i.e., local and global). To avoid the re-establishment of both the TCP connection and the OpenFlow channel, which is inevitable in conventional virtual IP address-based redundancy, we apply the multiple-controllers capability [9] to both local and global scenarios. Through the evaluation of the two scenarios, we use OpenFlow-1.2-compliant controllers and switches on Linux by extending an existing implementation [16] as shown in Section III.

A. Proposed Design of Local Recovery

First, we explain the redundant method in a single domain, which is typically a data-center hosting OpenFlow controllers. Table V shows parameters specific to a local-recovery experiment.

Fig. 6 shows a reference model for describing and evaluating the proposed scheme designed for the local recovery. OFC01 is connected to two controllers through two OpenFlow channels. In a normal operation, the role of OFC01 is set to MASTER and that of OFC02 is set to SLAVE. OFC01 and 02 have the same flow entry information mirrored between the two OFCs. OFS01 and OFS02 are

TABLE V. PARAMETERS SPECIFIC TO LOCAL-RECOVERY EXPERIMENT

Node	Parameter	Value
OpenFlow controller	keep-alive interval	50 ms
	keep-alive timeout	50 ms

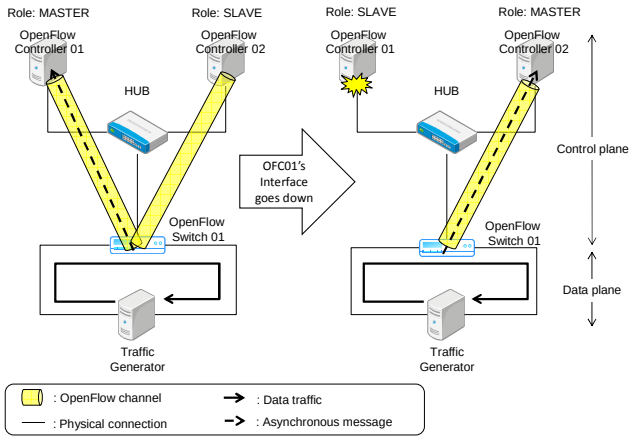


Figure 6. Experimental scenario using multiple-controllers capability in local environment

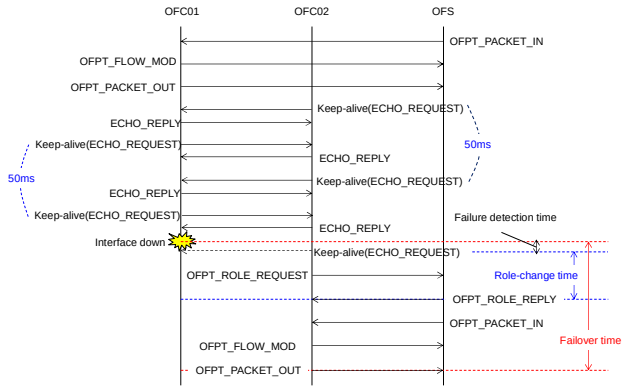


Figure 7. Design of a control procedure for a local recovery

operated under the reactive mode, and send a packet-in message to OFC01 when it receives a new packet undefined in the flow entry. To evaluate the performance influence in the data plane, a traffic generator continuously generates data packets with 100 packets per second (pps) where every packet has unique flow identifiers for stressing the reactive operation of the controller.

Fig. 7 shows an operational sequence of the proposed redundant scheme utilizing the multiple-controllers capability. In the proposed scheme, controllers send keep-alive messages (e.g., ICMP echo) to each other every 50 milliseconds. In a normal operation, OFS01 sends an asynchronous message such as packet-in to OFC01, since the switch recognizes the role of OFC01 as MASTER and that of OFC02 as SLAVE. OFC01 sends a flow-modification message and packet-out message to respond to the packet-in message from the switch. If the keep-alive message is lost, a controller (i.e., OFC01) is assumed to have failed. Due to the failure of OFC01, OFS01 cannot send any packet-in messages, and then the data plane cannot continue successful packet forwarding for any new incoming flows. Upon detecting the failure of OFC01, OFC02 sends an OFPT_ROLE_REQUEST message to OFS for changing its own role to MASTER. Then, OFS replies the OFPT_ROLE_REPLY message, and starts sending asynchronous messages to OFC02 after the completion of the role-change process. To respond to the asynchronous messages, OFC02 starts sending flow-modification and packet-out messages, and finally, the packet forwarding in the data plane is restored. As represented in Fig. 2, failover time is defined as the duration time from the failure event of OFC01 to the first packet-out message sent by OFC02. Failover time is measured using a traffic generator to obtain the data plane outage time. The role-change time is defined as the duration time from the detection of OFC01 failure to the receipt of OFPT_ROLE_REPLY by OFC02. Role-change time is measured by retrieving the event log of each controller to observe the control message process.

B. *Evaluation of Local Recovery*

The failover time and role-change time are evaluated by increasing flow entries in order to investigate the influence of the entry size. Fig. 8 shows the average of 10 measurements of the failover time and role-change time. Failover time is around 60-90 milliseconds and role-change time is about 15 milliseconds. Since the failure detection included in the failover time has a timing offset within the keep-alive interval, the observed failover time has some fluctuation range. Although the role-change time of the proposal is comparable with that of the virtual address-based redundancy, the failover time of the proposal shows a significant advantage thanks to the seamless handover between multiple OpenFlow channels. Fig. 8 also shows that entry size on OFCs does not affect the local recovery operation both for role-change time and failover time.

In the redundancy method that uses the multiple-controllers capability in the local recovery, we measure the CPU utilization of the NOX process due to failover. We use Open vSwitch as OFS instead of Ericsson TrafficLab 1.1 software switch. The traffic generator does not generate any data packet to measure only CPU utilization of the NOX process due to failover. Fig. 9 shows the average of 10 measurements of the maximum CPU utilization. According to Fig. 9, the CPU utilization of the NOX process increases with the growth of the number of OFSes. However, the utilization is smaller than that of using VRRP. This is because there is no process of OFS01's reconnecting (i.e., TCP reconnecting and OpenFlow reconnecting) to OFC02 in the proposed method of using the multiple-controllers capability. Thus, the proposed redundancy method of using the multiple-controllers capability has two advantages compared with the conventional method of using VRRP. First, its failover time is short because the process of failure detection is independent of the process of handover. Consequently we can combine the fast detection method (e.g., BFD [21]) with the

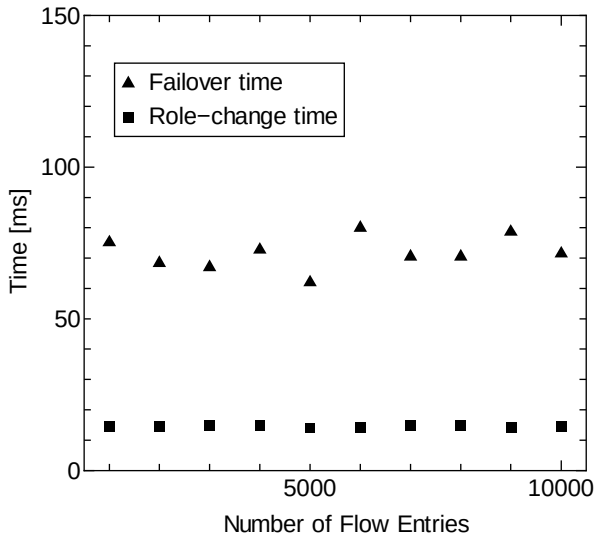


Figure 8. Result of failover and role-change time in a single domain.

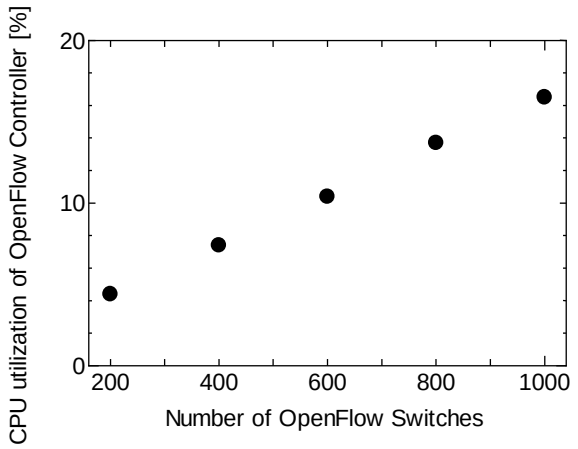


Figure 9. CPU Utilization of NOX process during multiple-controllers-based failover operation.

process of handover and we can achieve the short failover time. Second, considering that the CPU utilization due to the failover is low compared with the method of using VRRP, the proposed redundancy method of using multiple-controllers is suitable especially for a large OpenFlow network.

C. *Proposed Design of Global Recovery*

In this section, we explain the redundant method of multiple domains. Table VI shows the parameters specific to the global-recovery experiment. Fig. 10 shows a reference model of the controller redundancy for the global recovery scenario. The global recovery should consider tackling extraordinary events affecting, for example, the entire data center. We assume that a controller is installed in each domain to retain its scalability and performance. The controller manages OFSes belonging to the same domain as the MASTER, and the controller manages the other OFSes in the other domains as the SLAVE. The respective roles of the controllers are depicted in the upper side of Fig. 10. For example, OFS-A (i.e., some switches belonging to domain-A) recognizes the role of OFC-A (i.e., the controller belonging to domain-A) is MASTER and the role of the other controllers is SLAVE. Similarly, OFS-B and OFS-C also recognize the role of the controller that belongs to its same domain is MASTER and the roles of the other controllers are SLAVE. The controller has flow entry information for only OFSs recognizing the controller as MASTER. Thus, the controller does not need to have an excessive configuration or receive an excessive message. Additionally, one characteristic of our proposal is the existence of a Role Management Server (RMS). RMS monitors all controllers to manage their role, and RMS has some data such as CPU utilization, role information, configurations of all controllers and domain information of all switches. RMS determines which controller should take over the role of MASTER and relevant configuration data, if a controller has failed. In this regard, we have to be careful to prevent second failures. If OFC-B takes over the role of MASTER for broken OFC-A and places OFS-A under management besides OFS-B, there is the possibility of CPU utilization overload of OFC-B and then OFC-B may fail consequently. Thus, we should consider that one failure would induce subsequent failures. That is why RMS monitors CPU utilization and judges multiple-controllers should take over the role of MASTER from one controller, if RMS judges that taking over with a single controller raises overload of CPU utilization.

TABLE VI. PARAMETERS SPECIFIC TO GLOBAL-RECOVERY EXPERIMENT

Node	Parameter	Value
Role management system	operating system	Ubuntu12.04
	network interface	Gigabit Ethernet
	snmp monitoring interval	50 ms

The role of the controller

	OFC A	OFC B	OFC C
OFS A	Master	Slave	Slave
OFS B	Slave	Master	Slave
OFS C	Slave	Slave	Master

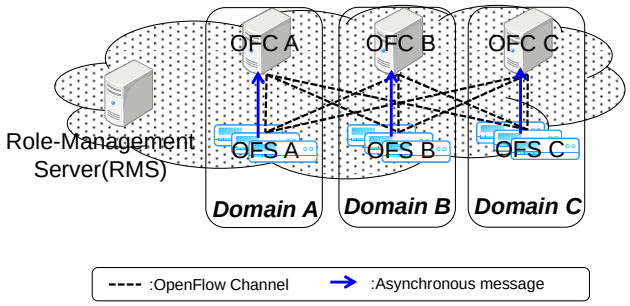
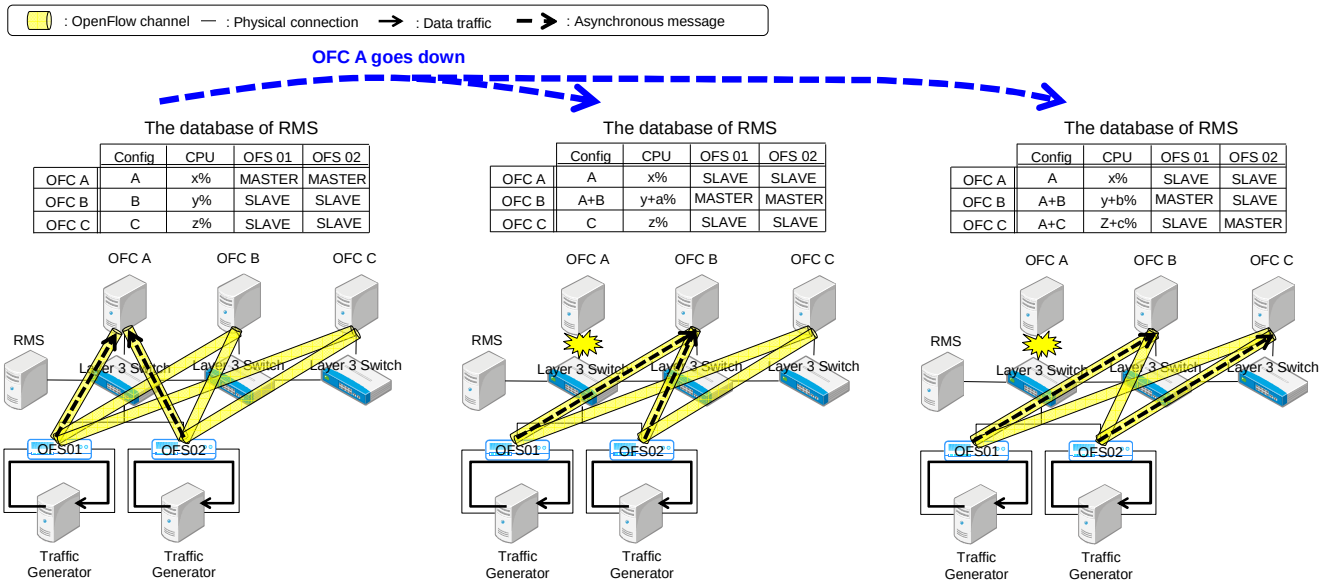


Figure 10. A network model for global recovery.



(a) Normal state.

(b) Two switches are migrated to a single controller.

(c) Two switches are migrated to two controllers.

Figure 11. Role-change transition in the global controller recovery

Fig. 11 shows the role-change transition for the global controller recovery. Fig. 11-(a) shows the initial state, and two switches are connected to three controllers through three OpenFlow channels. In the normal operation, both switches recognize that the role of OFC-A is MASTER and the other controllers are SLAVE. So only OFC-A receives some asynchronous messages such as packet-in messages. In this case, the three controllers have different configurations respectively and the information is reflected in the database of RMS. Also RMS has CPU utilization, the role information of each controller and the cognition haven by switch regarding the role of the controller in its database. The traffic generator connects OFS01 and OFS02 respectively and the data transfer rate is 100 pps. The two switches receive a new packet and send a packet-in message to the controller at all times as well as the measurement of a single domain.

If OFC-A fails and RMS judges there is no problem of a single controller taking over the MASTER role, the initial state (i.e., Fig. 11-(a)) is changed to Fig. 11-(b) where only OFC-B takes over the role of MASTER. The RMS database is updated accordingly, and both switches start sending asynchronous messages to OFC-B.

In contrast, if OFC-A fails and RMS judges that a single controller cannot take over the Master role but two controllers can, the initial state is changed to Fig. 11-(c) where two controllers take over the role of MASTER. The database of RMS is updated accordingly, and then OFS01 starts sending asynchronous messages to OFC-B. OFS02 sends asynchronous messages to OFC-C.

Fig. 12 shows a global recovery scheme in the case of Fig. 11-(b). RMS monitors the CPU utilization of all controllers every 50 milliseconds with Simple Network Management Protocol (SNMP) [22]. Since Fig. 5-(b) has three controllers, each controller is monitored every 150 milliseconds. The proposed recovery process consists of a judge-phase and a takeover-phase. If RMS is unable to retrieve the information about CPU utilization from OFC-A, RMS does not immediately assume that OFC-A has failed to avoid false positive. To ensure the failure detection, RMS requests that the ICMP echo be sent from the other controllers (OFC-B and OFC-C) to OFC-A. If more than half of the results indicate the failure of OFC-A, RMS determines that OFC-A has failed and starts calculating a new MASTER controller migrating OFC-A's configuration and OFSs under OFC-A. The process from failure detection to the determination of a failed controller is defined as the judge phase as indicated in Fig.12.

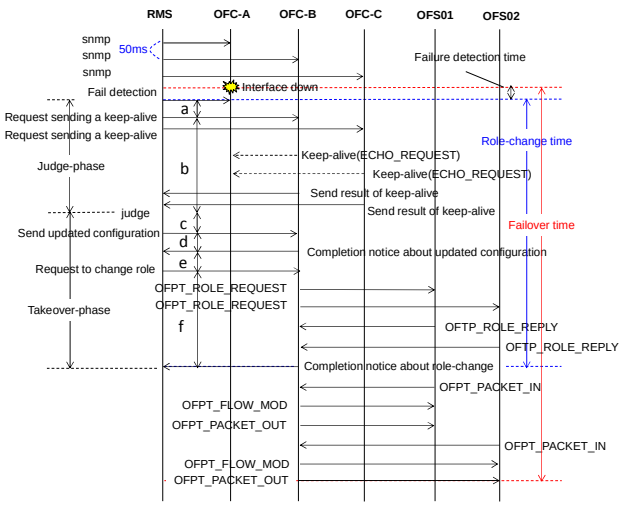


Figure 12. Proposed operational sequence for Figure 5 (b) scenario.

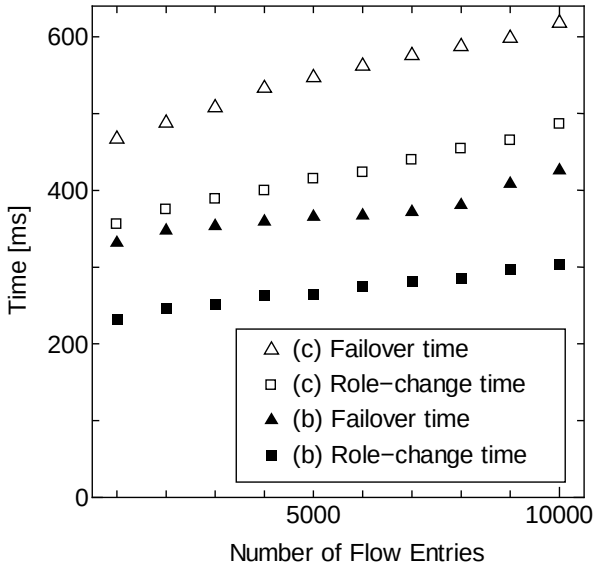


Figure 13. Result of failover time and role-change time in global recovery.

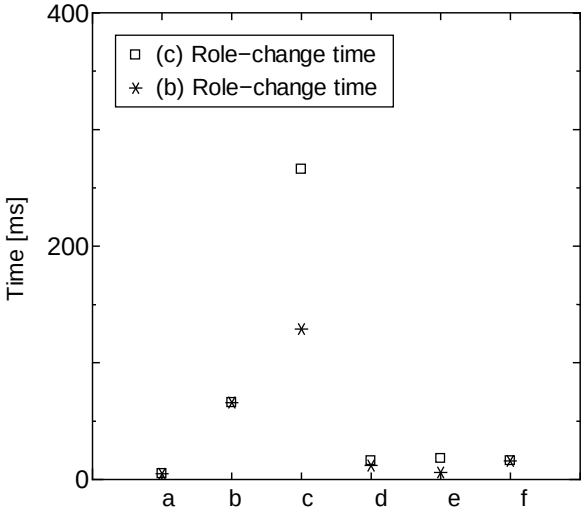


Figure 14. Breakdown of role-change time observed for scenario Fig. 11-(b) and (c)

After the judge-phase, RMS moves to the takeover-phase. In the takeover-phase, RMS firstly calculates whether it is no problem for a single controller to take over all switches connected to OFC-A by considering CPU utilization of OFC-A as well as OFC-B and C. If two or more controllers are required to take over all switches of OFC-A, RMS separates the switches based on the ratio of the available CPU resources of new MASTER controllers. If RMS decides that OFC-B is adequate to become a new single MASTER as shown in Fig. 11-(b), RMS integrates OFC-A's configuration into OFC-B's and registers the integrated configuration into OFC-B. Upon receiving the integrated configuration, OFC-B updates its own configuration and then reports the completion of the integration process. Then, RMS requests OFC-B to send the OFPT_ROLE_REQUEST to the switches for updating the role of OFC-A to SLAVE and OFC-B as MASTER. The switches send the OFPT_ROLE_-REPLY after updating the role change process. Then, OFC-B reports the completion of the role-change process to RMS. The process from completion of the judge-phase to completion of the role-change is defined as the takeover-phase. After the take–over phase, the switches OFC01 and 02 start sending asynchronous messages to OFC-B.

D. Evalution of Global Recovery

Fig. 13 shows the average of 10 measurements of role-change time and failover time in both cases of Fig. 11-(b) and (c). Role-change time and failover time increase with the growth of flow entry size. This result shows the difference in behavior compared with the result of a local recovery shown in Fig. 8. The major reason for this increase of failover time is that RMS needs integration of multiple configurations of failed OFC and registration of the configuration during the takeover-phase. As different scenarios of the global recovery, RMS selects multiple-controllers as the new MASTER as shown in Fig. 11-(c), and the scenario takes a longer role-change time and failover time as shown in Fig. 13. This reason is analyzed using the result of Fig. 14 that shows a breakdown of the role-change time under 1000 entries in both cases (i.e., Fig. 11-(b) and (c)). The characters ("*a*" to "*f*") placed on the x-axis of Fig. 14 correspond to the marker shown in Fig. 12. As shown in Fig. 14, the major performance difference comes from *c* that is the time to integrate configuration in RMS and register it to OFC. Current implementation suffers from the serial processing of the registration of integrated data. This means introducing parallel processing of the registration resolves the delay of role-change for the scenario shown in Fig. 11-(c).

According to Fig. 13, the role-change time is about 300 milliseconds and failover time is 420 milliseconds in 10000 flow entries, in the case of the scenario in Fig. 11-(b). In the case of the Fig. 11-(c) scenario, the role-change time is about 500 milliseconds and failover time is about 620 milliseconds. These results indicate that, for both scenarios, our proposal achieves a competitive role-change time and faster failover time compared with existing redundant mechanisms [13, 15]. We consider the proposed implementation of multiple-controllers achieves high availability controllers for both intra and inter data-center recoveries.

In this paper, we do not evaluate the redundancy of RMS itself. Although conventional server redundancy mechanisms accompanying a relatively longer failover time may be applied to RMS redundancy, RMS cannot be a critical bottleneck of processing asynchronous messages. This is because RMS failure itself does not affect any OpenFlow channel sessions and thus the data plane is not affected, accordingly.

V. CONCLUSION AND FUTURE WORK

In OpenFlow architecture, the controller is an important element for achieving reliable SDN. In this paper, we evalu-

ated the redundant method for the OpenFlow controller by using a conventional method (i.e., VRRP) and we verified the existence of the issue from the viewpoint of failover time and CPU utilization. And then we proposed a redundant scheme to tackle both a single domain ("local") and multiple domain ("global") recovery scenarios, which cannot be resolved with conventional redundant schemes. To avoid a long failover time and heavy CPU load due to conventional virtual IP address-based schemes, our scheme used the multiple-controllers capability for seamless handover. To avoid performance scale-limit due to conventional clustering schemes, our scheme eliminates any frontend server from the redundant system. The evaluation shows that the proposed scheme involves lower CPU utilization and competitive role-change and failover times compared with conventional schemes. In our scheme, the CPU utilization due to the process of failover is half or less compared with the virtual IP address-based scheme in the case of 1000 units of OFSes. Our scheme is more suitable for a large OpenFlow network. The role-change time observed in a local recovery scenario is about 15 milliseconds regardless of entry size, and that in a global scenario ranges from 200 to 400 milliseconds. CPU resource-aware migration of managed OpenFlow switches in the failover process was successfully achieved by our scheme. The proposal is expected to be an effective high availability scheme necessary for deploying reliable and scalable SDN.

In future work, we will shorten the failover time for the scenario of some OpenFlow switches migrated to some OpenFlow controllers. In RMS, we will separate the current redundancy process that is sequential migration into every controller, and we will establish CPU-based controller resource modeling to accurately handover many OpenFlow switches in the event of, especially, global recovery where massive nodes may need to be protected.

ACKNOWLEDGMENT

We are grateful to Yasunori Maruyama for our productive discussions, support for our experiments and programming assistance.

REFERENCES

[1] K. Kuroki, N. Matsumoto, and M. Hayashi, "Scalable OpenFlow controller redundancy tackling local and global recoveries," Proc. International Conference on Advances in Future Internet (AFIN2013), August 2013, pp. 61-66.

[2] N. McKeown et al., "OpenFlow: enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review, vol. 38, isssue 2, April 2008, pp. 69-74.

[3] M. P. Fernandez, "Evaluating OpenFlow controller paradigms," Proc. International Conference on Networks (ICN2013), January 2013, pp. 151-157.

[4] R. Pries, M. Jarschel, and S. Goll, "On the usability of OpenFlow in data center environments," Proc. IEEE International Conference on Communications (ICC2012), June 2012, pp. 5533-5537.

[5] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "OpenQoS: an OpenFlow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," Proc. Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC 2012), December 2012, pp. 1-8.

[6] "IEEE standard for local and metropolitan area networks – station and media access control connectivity discovery," IEEE Std 802.1AB, September 2009.

[7] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takacs, and P. Skoldstrom, "Scalable fault management for OpenFlow," Proc. IEEE International Conference on Communications (ICC2012), June 2012, pp. 6606-6610.

[8] A. Dixit et al., "Towards an elastic distributed SDN controller," Proc. ACM SIGCOMM Computer Communication Review, vol. 43, October 2013, pp. 7-12.

[9] "OpenFlow switch specification version 1.2," Open Networking Foundation, December 2011.

[10] "The transport layer security (TLS) protocol version 1.2," IETF RFC5246, August 2008.

[11] "Transmission control protocol," IETF RFC793, September 1981.

[12] A. Tootoonchian and Y. Ganjali, "HyperFlow: a distributed control plane for OpenFlow," Proc. the 2010 Internet Network Management Conference on Research on Enterprise Networking (INM/WREN'10), 2010.

[13] F. Koch and K. T. Hansen, "Redundancy performance of virtual network solutions," Proc. IEEE Conference on Emerging Technologies and Factory Automation (ETFA'06), September 2006, pp. 328-332.

[14] "Virtual router redundancy protocol (VRRP)," IETF RFC3768, April 2004.

[15] R. Zhang, T. F. Abdelzaher, and J. A. Stankovic, "Efficient TCP connection failover in web server clusters," Proc. IEEE International Conference on Computer Communications (INFOCOM'04), vol. 2, March 2004, pp. 1219-1228.

[16] CPqD/OpenFlow-1.2-Tutorial - GitHub, https://github.com/CPqD/OpenFlow-1.2-Tutorial [retrieved: April, 2013].

[17] NOXRepo, http://www.noxrepo.org [retrieved: April, 2013].

[18] TraffcLab/of11softswitch – GitHub, https://github.com/TrafficLab/of11softswitch [retrieved: April, 2013].

[19] Keepalived for Linux, http://www.keepalived.org [retrieved: October, 2013].

[20] Open vSwitch, http://openvswitch.org [retrieved: April, 2013].

[21] "Bidirectional forwarding detection (BFD)," IETF RFC5880, June 2010.

[22] "A simple network management protocol (SNMP)," IETF RFC1157, May 1990.