Scalable OpenFlow Controller Redundancy Tackling Local and Global Recoveries

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

***Abstract*—OpenFlow 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, and the high availability system should 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, the high-availability of the OpenFlow controller is investigated, and a redundant method considering both local and global (i.e., inter data-center) recoveries is proposed by using the multiple-controllers functionality that is defined in OpenFlow switch specification version 1.2 and later. The proposed redundant scheme eliminates frontend server causing limitation of performance scalability, while it achieves competitive role change and failover times.**

Keywords-OpenFlow; controller; redundancy.

I. INTRODUCTION

Towards future telecom services, programmability of the network is expected to shorten the service delivery time and to enhance flexibility of service deployment meeting diversified and complex user requirements on various applications (e.g., real-time and non real-time applications). OpenFlow [1] is an enabler of the centralized management solution meeting the aforementioned expectations, and many researches have addressed the scalability issue of the OpenFlow-based solution.

In [1], several OpenFlow controllers are evaluated from the viewpoint of scalability in centralized management and control. 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). In [2], the scalability of the OpenFlow solution for a data center environment is analyzed to show an implementation guideline. 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 an advantage of the centralized management, the OpenFlow controller should not be a simple flow switching policy server. In [3], OpenQoS architecture delivers end-to-end quality of service (QoS) with OpenFlow-based traffic control. The OpenFlow controller with OpenQoS has the role of collecting the network state to perform dynamic QoS routing, i.e., the controller has the route calculation function just like the Path Computation Element (PCE). Indeed, in 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 Software Defined Networking (SDN), and thus the high availability of the controller system must be discussed. However, there is little research on the high availability of the OpenFlow controller that must play the important role on SDN.

In this paper, the high availability of the OpenFlow controller is investigated, and a high availability method applicable to multiple OpenFlow controllers is proposed. In the proposed redundant method, "global" repair (i.e., inter data-center redundancy) as well as local repair (i.e., redundancy within a local network) are considered. The proposal achieves a competitive failover time compared with existent 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 explain the function of multiple controllers defined in OpenFlow switch specification 1.2 [5] and also explain its applicability to achieving redundancy of the OpenFlow controller. We investigate existent approaches of redundant schemes as well. In Sections III-A and B, we evaluate the performance of the redundant method for multiple controllers placed on a single domain. In Sections III-C and D, we propose the redundant method for multiple controllers placed on multiple domains and evaluate the performance. Finally, concluding remarks are given in Section IV.

II. BACKGROUND

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) [12] or a TCP connection [13] defined as a "secure 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

 ISBN: 978-1-61208-300-1

packet-in message over the secure channel to the controller in order to retrieve a direction on how to treat the packet.

One method handling data plane failure is implementing a monitoring function on OpenFlow switch [11]; however, only the monitoring function in a data plane is not sufficient to achieve high availability of the control plane. In contrust, achieving controller redundancy also contributes to protection of the data plane.In the case of the controller outages, the secure channel connection is lost accordingly, and then the packet-in message cannot be successfully processed by the controller. 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 never provides desirable services until the ultimate recovery of the controller.

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 has its own role by using the function of multiple controllers, 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 secure 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 secure channels, and thus the switch is not required to re-establish new secure 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, investigating the implementation of the controller side is important to achieve the redundancy; however, that has not been proposed yet. We use the capability of multiple controllers to achieve high availability of the control plane. In the following section, we propose how to use it and explain the effect.

III. PROPOSAL AND DEMONSTRATION

In this section, a proposed architecture for local and global recoveries is described, and recovery operation in the two scenarios (i.e., local and global) is demonstrated. To avoid the secure channel re-establishment that is inevitable in conventional virtual IP-based redundancy, the proposal commonly applies the multiple-controllers functionality [10] to both local and global scenarios. Through the demonstration for the two scenarios, we implemented the controller prototype based on NOX-C++ for OpenFlow 1.2 available in [10].

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.

Figure 1 shows a reference model to describe and demonstrate the proposed scheme designed for the local recovery. An OpenFlow Switch (OFS) is connected to two controllers through two secure channels. In a normal operation, the role of the OpenFlow Controller (OFC) 01 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. OFSs are operated under the reactive mode, and send a packet-in message to the controller 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.

Figure 2 shows an operational sequence of the proposed redundant scheme utilizing the multiple-controller capability. In the proposed scheme, controllers send keep-alive messages (e.g., ICMP echo) to each other every 50

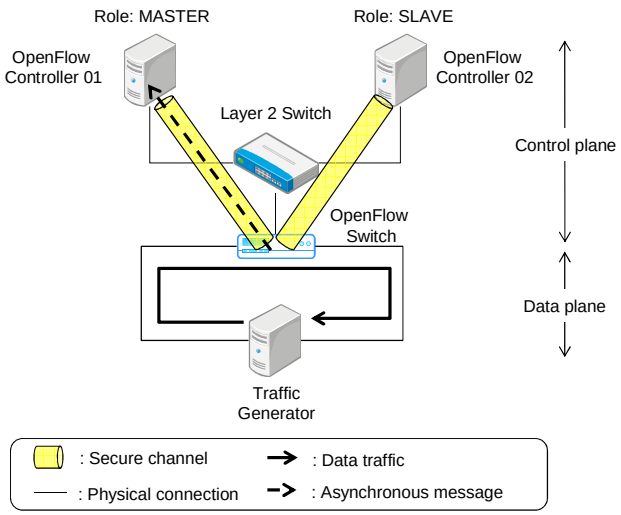


Figure 1. Testbed configuration for the local recovery.

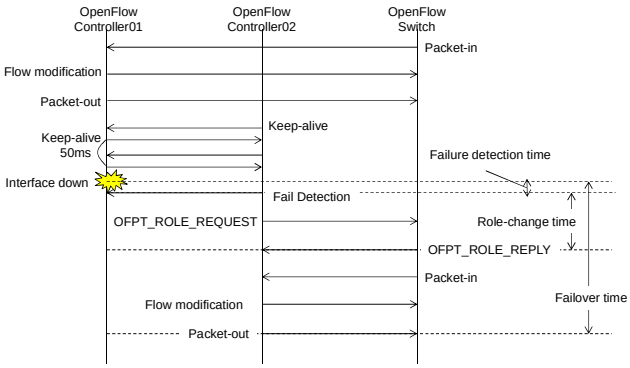


Figure 2. Design of a control procedure for the local recovery.

milliseconds. In a normal operation, OFS 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, OFS 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 Figure 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. A 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. *Demonstration of Local Recovery*

The failover time and role-change time are evaluated with increasing flow entries in order to investigate the influence of the entry size. Figure 3 shows the failover time and role-change time averaged with 10 times measurements. 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, 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 secure channels. Figure 3 also shows that entry size on OFCs does not affect the local recovery operation both for role-change time and failover time.

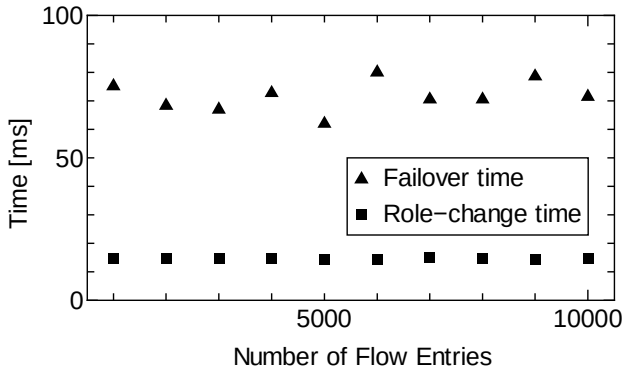


Figure 3. Result of failover and role-change time in a single domain.

C. *Proposed Design of Global Recovery*

In this section, we explain the redundant method of multiple domains. Figure 4 shows a reference model of the controller redundancy for the global recovery scenario. The global repair 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 OFSs belonging to the same domain as the MASTER, and the controller manages the other OFSs in the other domains as the SLAVE. The respective roles of the controllers are depicted in the upper side of Figure 4. For example, OFS-A (i.e., some switches belonging to domain-A) recognize 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, configuration 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 will 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

	Role of controller		
OFS A	Master	Slave	Slave
OFS B	Slave	Master	Slave
OFS C	Slave	Slave	Master

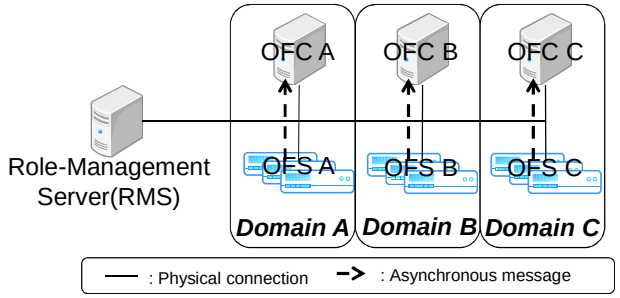
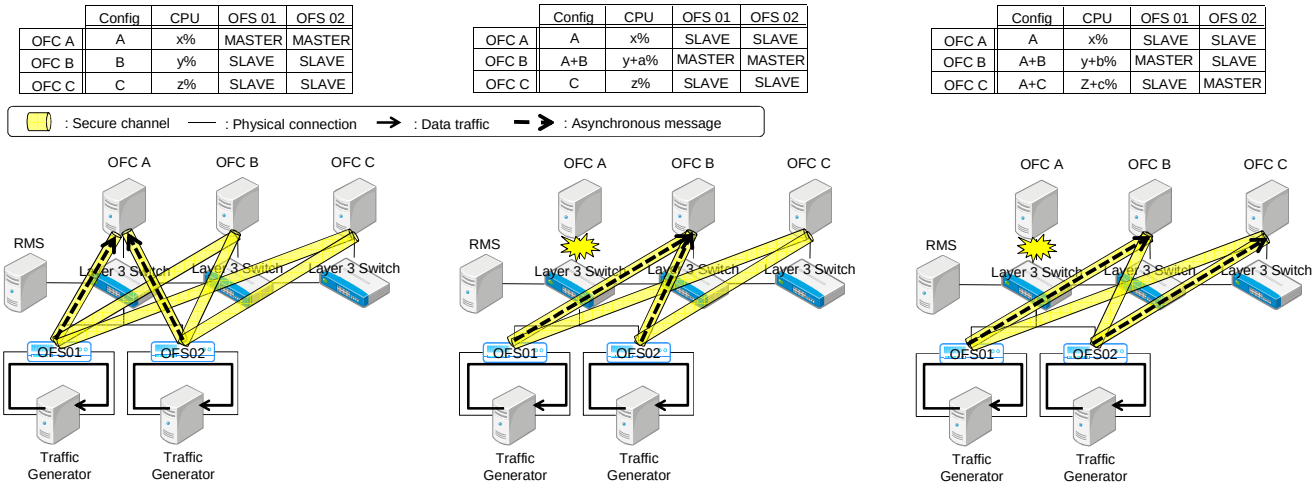


Figure 4. A network model for global recovery.



(a) Normal state.

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

(c) Two switches are migrated to two controllers.

Figure 5. Role-change transition in the global controller recovery.

that taking over with single controller raises overload of CPU utilization.

Figure 5 shows the role-change transition for the global controller recovery. Figure 5 (a) shows the initial state, and two switches are connected to three controllers through three secure 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, 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 to take over the MASTER role by a single controller, the initial state (i.e., Fig. 5 (a)) is changed to Figure 5 (b) where only OFC-B takes over the role of MASTER. The database of RMS is updated accordingly, and both switches start sending asynchronous messages to OFC-B.

In contrast, if OFC-A is failed and RMS judges that a single controller cannot take over the Master role but two controllers can, the initial state is changed to Figure 5 (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.

Figure 6 shows a global recovery scheme in the case of Figure 5 (b). RMS monitors the CPU utilization of all controllers every 50 milliseconds. Since Figure 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 Figure 6. 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

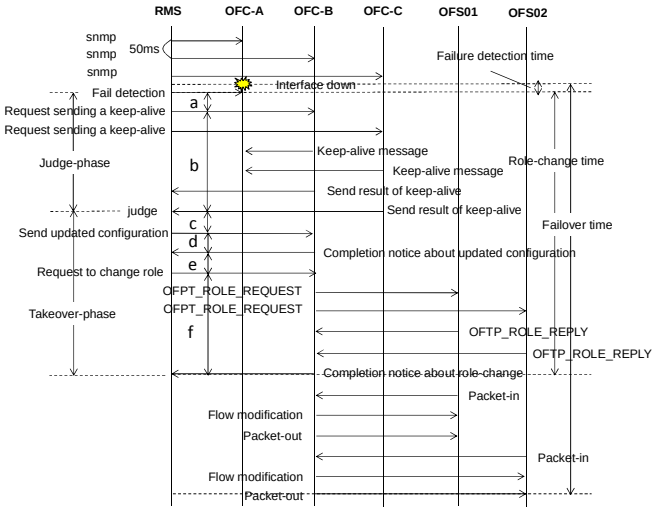


Figure 6. Proposed operational sequence for Figure 5 (b) scenario.

controllers. If RMS decides that OFC-B is sufficiently adequate to become a new single MASTER as shown in Figure 5 (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 OFPT_ROLE_REQUEST to the switches for updating the role of OFC-A to SLAVE and OFC-B as MASTER. The switches send 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. Demonstration of Global Recovery

Figure 7 shows the role-change time and failover time averaged with 10 times measurements in both cases of Figure 5 (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 Figure 3. 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 Figure 5 (c), and the scenario takes longer role-change time and failover time as shown in Figure 7. This reason is analyzed using the result of Figure 8 that shows a breakdown of role-change time under 1000 entries in both cases (i.e., Figure 5 (b) and (c)). The characters ("*a*" to "*f*") placed on the x-axis of Figure 8 correspond to the marker shown in Figure 6. As shown in Figure 8, 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 Figure 5 (c).

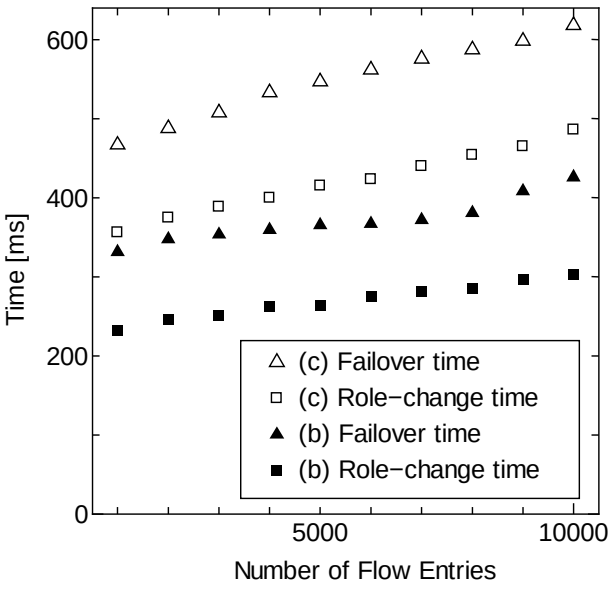


Figure 7. Result of failover time and role-change time in global recovery.

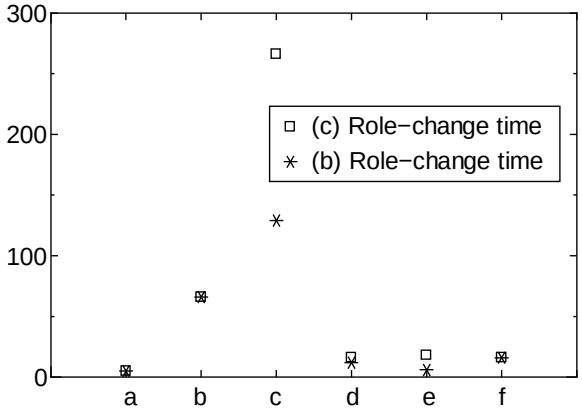


Figure 8. Breakdown of role-change time observed for scenario Figure 5 (b) and (c).

According to Figure 7, role-change time is about 300 milliseconds and failover time is 420 milliseconds in 10000 flow entries, in the case of the scenario in Figure 5 (b). In the case of the Figure 5 (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 competitive role-change time and faster failover time compared with existent redundant mechanisms [8, 9]. We consider the proposed implementation of multiple controllers achieves high availability controllers for both intra and inter data-center recoveries.

In this paper, we did not explicitly show the redundancy of RMS itself. Although conventional server redundancy mechanisms accompanying relatively longer failover time may be applied to RMS redundancy, single failure of the RMS itself does not directly affect packet forwarding.

IV. RELATED WORK

In [6], the HyperFlow 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 secure channel, and does not assume the multiple-controllers capability defined in OpenFlow 1.2. Therefore, time duration of the failover operation may increase with the growth of the number of switches managed by the failed

 ISBN: 978-1-61208-300-1

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. In [7], failover time is evaluated using the virtual address-based implementation with Common Address Redundancy Protocol (CARP), which is like Virtual Router Redundancy Protocol (VRRP) [8]. According to the analysis, the average time of changing the role between master and backup is 15.7 milliseconds. However there is a concern that the virtual IP-based approach takes a longer fail-over time than our approach, since the virtual IP-based approach fundamentally involves the re-establishment process of the secure channels. 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 repair. In [9], a server clustering with a mechanism of seamless handover of TCP connection between backend servers was proposed. While each TCP connection is visible to only one back-end server in a normal clustering scheme, the proposal [9] 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 repair involving multiple locations. However, from the viewpoint of performance scalability of the OpenFlow controller as analyzed in [1, 2], 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 our proposed solution, RMS cannot be a serious bottleneck of processing asynchronous messages because RMS failure itself does not affect any secure channel sessions and thus the data plane is not affected, accordingly. In addition, to tackle global repairs, server utilization should also be considered in the process of migrating many switches. However, conventional approaches do not consider utilization of the server resources (e.g., CPU).

V. CONCLUSION AND FUTURE WORK

In OpenFlow architecture, the controller is an important element to achieve reliable SDN. In this paper, 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 performance scale-limit due to conventional clustering schemes, our scheme eliminates any frontend server from the redundant system. The demonstration shows that the proposal performs competitive role change and failover times compared with conventional schemes. 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 is 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 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.

REFERENCES

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

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

[3] 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.

[4] 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), Dec. 2012, pp. 1-8.

[5] "OpenFlow switch specification version 1.2," Open Networking Foundation, Dec. 2011.

[6] 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.

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

[8] "Virtual Router Redundancy Protocol (VRRP)," IETF RFC3768, April 2004.

[9] 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.

[10] OpenFlow 1.2 Tutorial, https://github.com/CPqD/OpenFlow-1.2-Tutorial [retreived: April, 2013].

[11] 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.

[12] "The Transport Layer Security (TLS) Protocol Version 1.2," IETF RFC5246, August 2008.

[13] "TRANSMISSION CONTROL PROTOCOL," IETF RFC793, September 1981.

 ISBN: 978-1-61208-300-1