Using Adaptive Message Replication on Improving Control Resilience of SDN

Pang-Wei Tsai, Wai-Hong Fong, Wu-Hsien Chang, Chu-Sing Yang

Institute of Computer and Communication Engineering, Department of Electrical Engineering,
National Cheng Kung University, Taiwan
pwtsai@ee.ncku.edu.tw, fongwh1016@gmail.com, whchang@ee.ncku.edu.tw, csyang@ee.ncku.edu.tw

Abstract

Software-defined Networking (SDN) is considered a new solution in network provision for applying flexible functionalities. The characteristics of SDN include separated control and data planes, centralized management, having the global view of the network, fast adjustment, and adaptation. However, the centralized management brings resilience issues in the control plane of SDN, such as operation dependability, component survival, and behavior recovery. This paper introduces a resilience design for SDN to prevent network behavior loss when the active controller fails. The aim of this paper is to develop a way to improve the resilience of SDN control message processing during the controller switching procedure. The proposed design has been practiced on RYU controller for verification and evaluation. The initial performance tests show that our work is able to increase the reliability of the controller at the expense of lower performance.

Keywords: Software-defined networking, Controller, Reliability, openflow, Resilience, Replication

1 Introduction

The prototype of computer networks consisted of the telegraph lines that sent instructions to the computers. Designated computer networks became an independent source of connectivity [1]. With the geographic expansion of networks, the internet was created [2]. For fulfilling connectivity requirements in a network system, routing and switching protocols are used to control how packets are delivered. By exchanging control messages among linked neighbors, the network node can learn about the adjacent status and determine a feasible forwarding paths [3-4]. For improving flexibility of the control, in future networks, the so called Software-Defined Networking (SDN) [5] is recognized as an innovative architecture [6]. The SDN protocol is defined as an open standard, with vendor-independence. This reduces proprietary hardware dependence on certain manufacturers, performing network nodes work together under open and integrated management.

In recent years, SDN protocols, like OpenFlow [7], have been applied on many networking fields, including data centers, WAN communications as well as routing exchanges. Some networking device manufacturers also proposed their solutions [8] for enabling SDN in their products. The significant characteristics of SDN are its centralized control and global view. These two characteristics bring several advantages to network operation. For examples, the centralized control makes service appliances to be extended easily, and optimization mechanism like traffic engineering algorithm can be efficiently operated owing to the global view [5]. However, the reliability of SDN is problematic. Since the network nodes in SDN are not running autonomously, the dependence on the controller increases the vulnerability of control communication. If interruption occurs in the control channel, the communication between the controller and the network node breaks down as well. The controller no longer receives the latest network status from the nodes, neither sends new instructions to manage them how to deliver the new incoming traffic flows. Once the controller malfunctions, it may cause no more request in the control plane can be handled, leading packets of new flows to be dropped in the data plane.

Due to the above reasons, the reliability of SDN became a popular research topic [9-11] recently. For achieving adequate protection, control messages in SDN control plane should be protected and executed correctly. When controller failure occurs, the spare controller must take over the leadership rapidly, and network nodes migrate their control connection to the spare controller. During the failover procedure, the difficulty of the procedure is to make sure that network control actions are continuously registered. Figure 1 shows an example of the control plane events during the controller failover. When the procedure of controller switching starts, control messages may get lost at this point. Before a network node notifies of the

*Corresponding Author: Chu-Sing Yang; E-mail: csyang@ee.ncku.edu.tw
DOI: 10.3966/160792642018121907019

new controller instance, it keeps sending query messages to the failed one. The un-processed control messages in the failed controller are usually discarded. Hence, how to prevent network behavior loss when switching controller becomes one of research issues in SDN reliability.

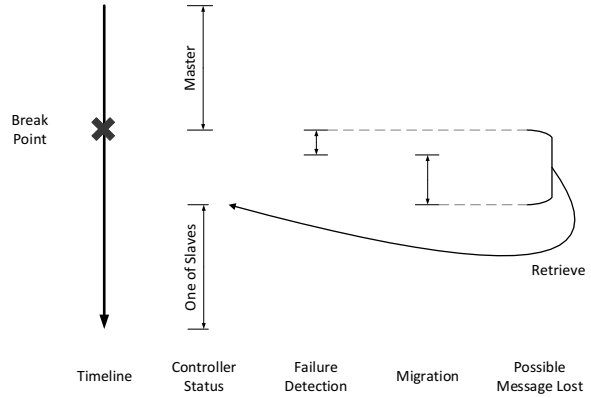


Figure 1. An example of control plane influence during the failover procedure

In this work, we propose a prototype solution to make improvement on protecting control messages execution. The solution is applicable on an SDN network equipped with spare controllers, and it aims to support the controller switching by keeping the network status uninterrupted. In our reliability development, each control message sent from the SDN switch is replicated, and a procedure is responsible to ensure that the message has been received by all joined controllers. For controller failure detection, a heartbeat immigration control is implemented, and there is also an adjustment mechanism applied to make the operation adaptive and efficient. When controller failure is detected, our reliability development tries to tag un-processed messages on spare controllers, and it makes sure that these messages to be retrieved when the selected spare controller takes over the control responsibility. By doing this, we can reduce the chance of missing control messages for the controller system and lower down the possibility of restarting the reactive flow rule installation on the switch, improving the control reliability of SDN.

The remainder of this paper is organized as follows. Section 2 has a brief review of related works. Several fault-tolerance solutions applied for SDN controllers are introduced. Section 3 shows our proposed method, explaining system design and development details. The experiments for verification and evaluation are described in Section 4, and a little case study is also presented. Finally, the conclusions and future work are given in Section 5 and Section 6.

2 Background and Related Work

This section introduces the basic concepts of SDN, and explains some aspects of reliability design as well as implementation for improving the reliability on it.

2.1 Software-Defined Network and its Attributes

Comparing legacy network with SDN architectures, the most significant difference is the form of control. The legacy network usually uses in-band control, and network nodes negotiate with each other to determine the forwarding rules. In contrast, the SDN uses out-of-band control, and each network node in SDN is basically managed by the controller. To illustrate their differences, an architecture comparison is shown in Figure 2. The SDN architecture [6] can be divided into three parts: application plane, control plane and data plane. The application plane runs as an entrance for integrating network applications. The applications use Northbound Application Programming Interface (API) to send the requirements to the control plane for changing the network behavior. The control plane is the essence of the network operation and management. The control logic in this plane is responsible for receiving the requirements and turning them into control messages for the network nodes. In data plane, the network nodes follow the instructions and forward the packets according to the instructions made by controller.

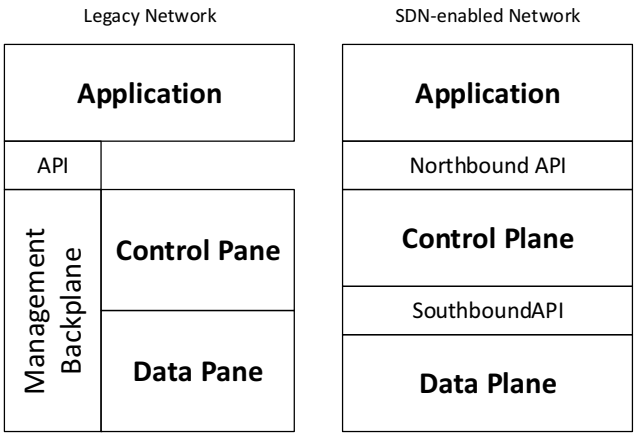


Figure 2. The comparison between the control architectures of legacy and SDN-enabled networks [6]

By separating the control logic from the data processing unit, SDN enables programmability on the network. Network operators can arrange network resources in a more flexible manner. Operations such as packet forwarding, filtering and duplication can be done easily through software-driven methods. For example, OpenFlow is a practical implementation of SDN-enabled networks in research and education [12-13]. It provides a specific design for SDN components, including the flow table and the control channel [14]. The flow table consists of data flow entries used for matching and forwarding packets. When mismatch occurs at a new arrival packet, this packet is sent to the controller for making forwarding decisions. After the controller decides the corresponding actions for the

packet, it sends the control message back to the network node with a new flow rule for packet transmission.

2.2 Fault-tolerance Issues in SDN

The fault-tolerance is the ability of self-adaptation to dynamic environment conditions in the network, including malicious attacks, operational overload and misconfiguration [15-16]. Silva et al. [17] investigated related problems in SDN operation. In their survey, the problems can be categorized as follows: survivability, traffic tolerance, disruption tolerance, dependability, security, and performability. Survivability is the ability to address a small number of random uncorrelated faults. Traffic tolerance enables unusual but legitimate traffic loads without interrupting operation. The dependability is the measured availability of the network. In security problem, it is related to the protection from harmful operation that affecting the system. The Distributed Denial of Service (DDoS) is a kind of security threat as seen in Yan et al. [18]. In the last, the performability is about the working efficiency in operation.

In SDN controller reliability, based on our observation [16-17, 19-20], there is an emerging trend to implement the control plane in a physically distributed but logically centralized architecture. Due to the control plane in SDN is centralized, implementing failover methods can make the probability of the control decision interruption to be minimized when control plane failure occurs. To introduce the reliability designs applied on SDN control framework, several typical developments for enhancing the ability of making control decisions without interruption are listed as follows:

2.2.1 A Replication Component for Resilient OpenFlow-based Networking

Fonseca et al. [21] proposed a primary-backup mechanism that manages the data structure, by relating a MAC address with a switch port. The messenger component provides synchronization communication between controllers. To check the liveness of controllers, the connected switch sends an inactive probe periodically. When there is a fault, the switch connects to one of the backup controllers.

2.2.2 ONOS

The ONOS [20] is an SDN control system that integrates controller instances with database and management service to support core network level traffic engineering. The main design purpose of ONOS is building a high performance SDN controller. It is capable of tolerating the failure of a controller by detecting the contacts with other controller instances. The ONOS provides a cluster control integration with distributed controller instances. However, the ONOS is not considering the communication lost among the switch and the controller instances natively. For the switch, only the control message successfully received by its primary controller will be handled.

2.2.3 RAVANA

Katta et al. proposed RAVANA [22], a fault-tolerant control plane that guarantees controller messages to be handled exactly once by an extended switch side through the OpenFlow interface. The design of RAVANA assumes that the behavior of controllers is deterministic, and the unmodified control application can be applied to them. It aims to replicate state-machine with lightweight switch-side mechanism to guarantee correctness of switch control. In their implementation, there is a reference deployment showing the ability of RAVANA for enabling unmodified controller applications to be executed in a fault-tolerant case.

2.2.4 ResilientFlow

To protect SDN-enabled networks under large-scale, unexpected link failures, Omizo et al. proposed ResilientFlow [23] to improve reliability for the SDN system. They developed a module called Control Channel Maintenance Module (CCMM) to detect control channel failure. In ResilientFlow, all switches of SDN have to maintain their own control channel, and it secure the SDN control plane when control communications between switches and controllers are failed.

2.2.5 Scalable OpenFlow Controller Redundancy Tackling Local and Global Recoveries

Kuroki et al. [24] proposed a mechanism to enhance the controller redundancy in OpenFlow. In their research, the high-availability of the OpenFlow controller is investigated, and two actions (i.e., local recovery and global recovery) in the fault recovery operation are designed. The demonstration shows that the OpenFlow switches in the failover process are successfully achieved failover operation.

2.3 Discussion

With the rising trends of implementing cloud-based techniques, the legacy network architecture may have insufficient flexibility to fulfill innovative requirements. Some non-regular network configurations are able to be implemented for achieving alternative control. However, even the SDN provides more software-defined functionalities to manage the network, improving its reliability is still a challenge. The possible issues for developing SDN control resilience are summarized in Table 1. The disruption tolerance is commonly used to make improvement on weak and

episodic connectivity. Its approaches include error correction schemes, multi-path routing, flow migration, and store carryforward schemes. Dependability ensures the service reliability of a system. It focuses on availability, safety, integrity, and maintainability of services. Security deals with unauthorized access to a network. In contrast to availability and dependability, it deals with the information assets instead of services. Finally, performability is the metric used to evaluate the performance of the network. Path selection and Quality of Service are two instances expressed in this property. Furthermore, to provide more flexibility and adaptation on reliability control, applying abstract control mechanisms, like NFV [25-26], will introduce more complicated functions as well as more non-deterministic operation to the controller.

Table 1. Instances of reliability research in SDN control

Research issues	Approachment	Related discipline	Remarks
Component Failure	Hardware Redundancy / Placement Optimization	Dependability	The spare components are able to take over the system operation when hardware failure occurred.
		Survivability	The placement optimization provides flexibility and adaptation for maintaining the spare infrastructure.
Operation Failed	Software Debugging / Fault-tolerance Technique	Dependability	Automation trouble shooting and examination are able to help network operators to determinate the possible problems.
		Survivability	The fault-tolerance techniques are often implemented on control plane for dependability, keeping system working consistency.
Optimization	Traffic Engineering	Disruption Tolerance	The link redundancy, path selection, load balancing, QoS and other policy-based management are the issues for implementing traffic engineering.
		Performability	The improvement on packet processing makes better performance and quality on packet transmission with high availability.
Measurement	Monitoring	Dependability	The probe detection provides network state in a given instant for checking property.
		Traffic Tolerance	The traffic stats are able to use for anomaly detection in management.
Security	Packet Deception / Traffic Log Analysis	Security	Securing the connection between the controller and switch can compromise the integrity and confidentiality of the control communication.
		Traffic Tolerance	Mitigating the threats such as DDoS attack and ping of death are necessary for network protection.

In most cases, implementing control resilience in the control plane increases reliability, however, system performance is a trade off [19]. Applying data synchronization on control logic is common used in distributed controller designs like ONOS. In this way, the performance of the control plane depends largely on the performance of the data synchronization. The data structure and the type of the stored data influence the speed of writing and reading the data correctly. Therefore, in order to increase the performance of controller synchronization, ONOS has changed the data structure. A different approach is to replicate and examine control messages [27]. The typical design in this way is to use a state-machine like RAVANA. However, its replication might not be able to guarantee deterministic outcome if there are numerous policy-control modules built in controller. On the other hand, the ResilientFlow is focusing on switching the control communication from the failed controller to the spare one and keeping network traffic forwarding well, not to make examination on possible missing messages during the controller handover. Moreover, considering the spare controller changes its role from slave to active for taking network control [24], the network behavior is not synchronized in local recovery. The new active controller may have no information of legacy network behavior (e.g., flow and port information). The chance that causes inconsistency in this way is greater than our development. Furthermore, in global recovery, even the role management server is able to keep copies of control information, the amount of updating data (like flow entry) in recovery procedure makes influence on mean time to recovery. Therefore, based on above aspects, instead of a replicated state-machine, we would like to opt for primary-backup replication in the control plane and develop soft-warized methods to make examination on control messages for improving SDN reliability.

3 System Design

To explain our resilience design built to improve SDN control reliability, in this section, we introduce the basic idea and mechanism design first. The system architecture, development and implementation are also described to let readers realize our idea and practice.

3.1 Basic Idea

To determine the possible ways of making reliability improvement in the SDN control plane, the investigation of component operation is necessary. In SDN, the control communication between components (e.g., switch and controller) follows the client-server model [5]. For example, when a new arrival packet is mismatched on the flow table, the switch sends variation (i.e., PACKET_IN message) to the controller. After that, the controller tells the switch how to deal with this packet. Furthermore, when any flow rule is expired or the port status turns down, the corresponding messages (i.e., FLOW_REMOVED and PORT_STATUS) are sent by switch or controller to make the notification.

For illustrating the above operation, we roughly organize the switch-controller interaction as a chain, shown in Figure 3, and the failure probability on each step is also included. In this figure, the meaning of each element are listed as the following:

A Triggering Event (switch)
B Query (switch to controller)
C Making Decision (controller)
D Rule Update (controller to switch)
F_q The probability that query fails for one controller
N_{ctr} The number of available controllers for switch
$F_{q.all}$ The probability that query fails for all controllers
F_{ctr} The probability that decision making causes failure on the controller
$F_{ctr.r}$ The probability that failure causes decision making retry on the controller
$F_{ctr.t}$ The probability that failure causes switch-and-controller interaction do termination

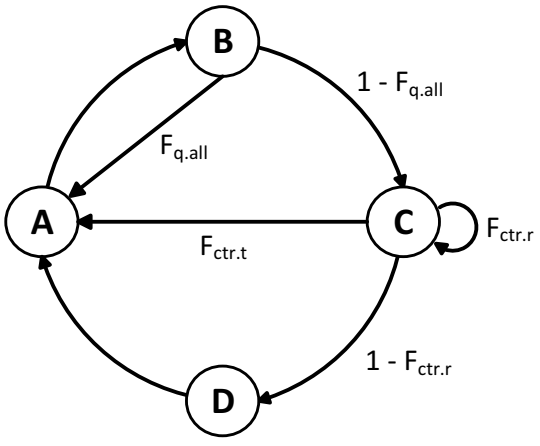


Figure 3. The transition diagram of control communication among SDN components

In the initial status, a detected event triggers the switch to make contact with the controller for asking how to deal with the situation. On the next status, the switch tries to send a query message to the controller for finding out the solution for the corresponding event. When there are multiple controllers ($1 < N_{ctr}$) connected to the switch, the switch will ask each controller according to query priority (like master and slave controller configuration of OpenFlow switch). If all controllers fail on the attempt, the query is terminated, and the status returns to the beginning. The probability of this situation ($F_{q.all}$) is formulated in equation (1).

$$F_{q.all} = (F_q)^{N_{ctr}} \quad \textbf{(1)}$$

After the controller receives a query message, it is responsible for telling the switch how to deal with the event. In decision making step, there is a chance ($F_{ctr.r}$) which the controller retry to make decision when the fault in the controller is recoverable. While if the un-recoverable fault or control message lost happens, the switch-and-controller interaction will be terminated $F_{ctr.t}$. The overall failure chances (F_{ctr}) in this step can be calculated by equation (2).

$$F_{ctr} = F_{ctr.r} + F_{ctr.t} \quad \textbf{(2)}$$

The last step in the operation is rule update. In this part, the controller sends instructions to let the switch realize how to deal with the packet. No matter whether the update action succeeds or fails, the status returns to the initial one in the end. Eventually, when update action is failed, the above process will start over again later owing to the event triggers query operation again.

To enhance the reliability of the controller, it is a common way to add many controllers into the control plane, and setup switches to link with these controllers. For instance, OpenFlow switch is able to setup one master controller and several salve controllers in operation. Once the master controller has no response in a period of time, the switch starts to query the secondary controller (role change procedure). This way is able to reduce the chance of failure ($F_{q.all}$) happening in query step. Furthermore, in decision making step, implementing controller collaboration to secure the control logic is also available to reduce the chance of decision failure ($F_{ctr.r}$). However, as mentioned in previous section, there is a probability that some reasons cause decision making failed. For one, the synchronization may not perfectly replicate all query messages due to the timing gap. For another, when controller switching, the migration time (even it is a very short time) may lead control logic malfunctioned temporarily. Moreover, to reach graceful handover, it is necessary to examine the un-processing control messages for avoiding control message lost. Therefore, we propose a reliable design to get query messages from the switch and supervise the decision making in multiple controller environment. When controller failover happens, our reliability development makes handover controller process the replication control messages, reducing the chance ($F_{ctr.t}$) of switch-and-controller interaction.

3.2 Mechanism Design

In development, our key point is to preserve the changes made by executed SDN control messages

continuously. When controller switching happens, the last known operation on the broken controller is expected to be handled by the handover controller. For achieving this, we proposed a mechanism with four functionalities to enhance the control reliability.

3.2.1 Replication Model

There are two typical replication schemes as described: passive and active [27]. Active replication is used in a state-machine. It requires all non-faulty components to receive commands in the same order. On the other hand, passive replication designates a unit to be the primary handler, and it is responsible for sending replication to the others. Since SDN operation relies on the control communication heavily, using active way may cause more loading on the controller then passive one. Therefore, in replication model, we choose to use passive way in our design, duplicating the control messages and sending them to spare controllers. For each spare controller, it keeps receive the resilience message while skip make decision for them.

3.2.2 Control Message Examination

The biggest difficulty to enhance controller reliability is to preserve the status of the network from time to time. Therefore, we have to supervise the progress of decision making. For ordinary spare controllers, to determine which replication message has been processed by active controller is necessary. Therefore, we would like to make active controller send pointer for notifying progress status in message examination.

3.2.3 State Recovery

When failure occurs, two aspects of controller fault-tolerance are required. One is to transform control authorization from the failed controller to a functional one. In this part, the performance heavily depends on the controller framework and hardware, and its improvement is out of the scope of our work. Another aspect is to retrieve controller operation as quick as possible since control messages are time-sensitive. For this aspect, according to the preserved pointer information, the handover controller is able to start decision making at the last known un-processing message, and the network is expected to keep its status continuous.

3.2.4 Extensibility for Development

Owing to one of characteristic in SDN is its separated control plane and data plane, the failover mechanism should be able to operate without hardware dependence on specific manufacture devices. In ideology, the reliability design should have open and extendable architecture for inserting further development modules. Owing to Ryu [28] and OpenvSwitch (OVS) [29] are two common OpenFlow-compatible solutions for satisfying above points in academic use. Our work in this paper is based on them to make system development and implementation.

3.3 System Architecture

In our design, four new modules are added to the Ryu controller, which are called: Switch Manager, Request Buffer, Inter-Controller Communication, and Failure Detection module. According to the controller's roles in OpenFlow protocol, the Active (i.e., master controller) and Standby (i.e., slave controller) modules may have different tasks and actions. The designed controller architecture is shown in Figure 4, and the details of each modules are described below:

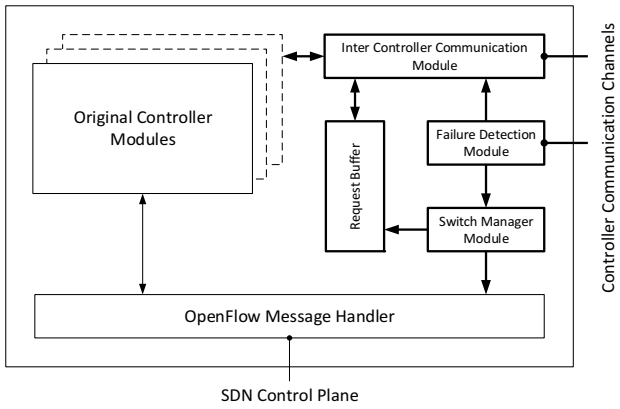


Figure 4. The architecture of designed modules in the controller

3.3.1 Switch Manager Module

The Switch Manager module is making contacts with SDN switches. After the switch establishes a control tunnel with the controller, the switch exchanges the Hello Message first, and the controller sends a Switch Feature Request and a Port Description Message to acquire feature and port information from the switch. Afterwards, the Switch Manager module makes negotiation with each switch according to the role of the controller. The Set-Asynchronous function lets all controllers receive every control message. It enables the switch to duplicate outgoing control messages, and send them to both the Active and Standby Controllers.

3.3.2 Request Buffer

Considering the performance requirement for duplicating control messages, we made a modification on OpenvSwitch to make request copies to all controllers. These messages are received by the OpenFlow Message Handler first. The Request Buffer module provides a buffering space for the system. For

the Active Controller, the request will be processed; for the Standby controller, the received control messages will be preserved here, waiting for the examination process.

3.3.3 Inter-Controller Communication Module

The Inter-Controller Communication module (ICC module) is responsible for coordinating the process of exchanging messages between the Active Controller and Standby Controllers. The controllers use their ICC modules to negotiate the asynchronous examination procedure. Due to network transmission, there is timing difference between the messages received by different controllers. When a control message is stored in the Request Buffer of the Active Controller, the ICC module reads it, and then the ICC module sends its content with identification information to the Standby Controller. The Standby Controller then searches its Request Buffer and examines the content. After that, the Standby Controller responds to the message with a confirmation. For cost-efficiency, we added a batch support function to the ICC module. It allows operators to setup a batch size during message forwarding. The batch process is shown in Figure 5. By doing so, the transmission frequency of the ICC module can be lowered when there are many incoming messages. Unfortunately, this function reduces control resilience in the system. For achieving a graceful protection, we recommend to let the ICC module forward any incoming message as soon as possible after examination.

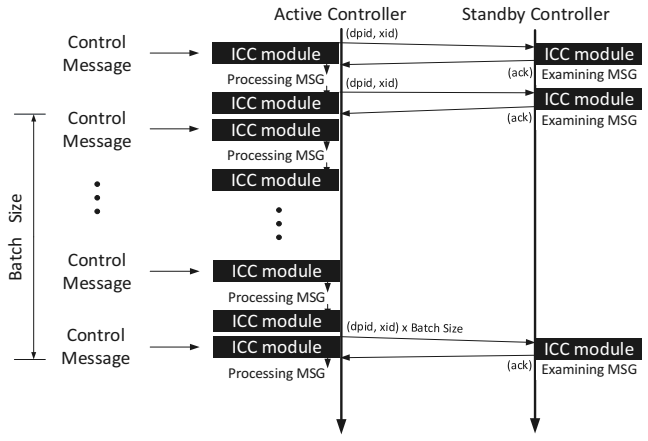


Figure 5. The batch process between the Active and Standby Controllers

3.3.4 Failure Detection Module

For detecting the liveliness of the controller, this module is designed to carry out heartbeat detection. In operation, the Active Controller periodically sends a heartbeat message to advertise its liveliness to the Standby Controller. Since the Ryu controller does not support real time processing, the clock synchronization problem might cause inaccurate heartbeat advertising. For overcoming this problem, the Failure Detection module in the Standby Controller is designed to update the timestamp token by averaging the elapsed time between two received heartbeat messages. For operating heartbeat mechanism adaptively, we put the average delay [30] concept into the heartbeat diagnostic. The heartbeat development is shown as Algorithm 1 and Algorithm 2. Since the Ryu controller is an event-triggered architecture natively, the developed heartbeat reaction can be activated by corresponding event. Considering the time variation, the expected timestamp of next received heartbeat on Standby Controller will be adjusted alternatively. When the Standby Controller does not receive heartbeat from the Active Controller in expected time, it will mark up this event and keep waiting for the next expected heartbeat. Once the Standby Controller misses two consecutive heartbeats, it suspects the Active Controller failed, and activates controller failover operation.

Algorithm 1 Heartbeat – Active Controller

```
1: Event_init //Receiving an event during initialization
2: Event_timeout //Receiving an event when timeout
3: sendHeartbeat ( ) //The function used to send the heartbeat message
4: L_sc //The list of Standby Controller
5: T_init //The default waiting time for sending next heartbeat
   message (the suggestion value is ≤ 1 second)
6: startTimer ( ) //The function used to setup the heartbeat timer for
   triggering Event_timeout
7:
8: procedure NEWEVENT(Event_init || Event_timeout)
9:     sendHeartbeat(L_sc)
10:    startTimer(T_init )
```

Algorithm 2 Heartbeat – Standby Controller

```
1: Event_init //Receiving an event during initialization
2: Event_heartbeat //Receiving an event when heartbeat coming
3: Event_timeout //Receiving an event when timeout
4: T_init //The default waiting time for sending next heartbeat message
   (it should be setup as same as the vaule on Active Controller)
5: T_last //The arrival timestamp of last heartbeat
6: T_current //The current timestamp
7: T_expect //The expected timestamp of next heartbeat
8: update Timestamp() //The function used to return timestamp for
   updating T_current
9: startTimer() //The function used to setup the heartbeat timer for
   triggering Event_timeout
10: dirtybit //The control flag used to mark the heartbeat missing
11: triggerHandover() //The function used to trigger controller handover
12:
13: procedure NEWEvent(Event_init)
14:     T_current = updateTimestamp()
15:     T_expect = T_current + T_init
16:     T_last = T_current
17:     startTimer(T_expect)
18: procedure NEWEvent(Event_heartbeat)
19:     T_current = updateTimestamp()
20:     T_expect = T_Current + (( T_Current - T_last ) + (T_expect - T_last)) / 2
21:     T_last = T_current
22:     if (dirtybit) then
23:         dirtybit = false
24:     startTimer(T_expect)
25: procedure NEWEvent(Event_timeout)
26:     if (dirtybit) then
27:         triggerHandover()
28:         break
29:     else
30:         dirtybit = true
31:         T_current = updateTimestamp()
32:         T_expect = T_current + (T_current - T_last )
33:         startTimer(T_expect)
```

3.4 Control Message Examination

Because of the network translation latency, the information delivery has timing difference. For failover control communication, our design replaced a small part of the data structure in OpenFlow protocol. In regular control communications, the Transition ID is a field in OpenFlow protocol [32] that is originally used for a controller-to-Switch response message attached to the corresponding request message. Moreover, the controller uses the Transition ID to identify the response of each Controller-to-switch message to avoid an unexpected order in command execution. In our development, the Transition ID has been re-organized by attaching a serial number to each OpenFlow control packets, letting controllers identify the order of received control messages timely. On the other hand, supervising all control messages will achieve a complete protection, while it costs much more resources and degrades the system performance. Focusing on the control messages which changed the network status is a cost-efficient way for balancing system overhead and availability. It is optional to replicate non-essential messages (like Hello Message) in the control plane. Accoring to above reasons, we suggest to track following events mainly: flow modification, port status change, and topology change. The details of corresponding control messages are listed in Table 2.

Table 2. The selected control messages for control plane resilience

Type	Control Function	Important Arguments	Event	Remarks
Flow	flow_removed	cookie, table_ id, match	flow modification	A flow is removed (owing to expiration or instruction).
Port	addPortState	dpid, openflow_ msg_raw	port status change	An available port is added on the switch.
	removePortState	dpid, port number	port status change	A port is removed on the switch.
	modifyPortState	dpid, openflow_ msg_rawr	port status change	The port status has been changed (owing to exception or instruction).
Port Status	set_downPortDataState	dpid, port number	port status change	A port is set to down on the switch.
	del_portPortDataState	dpid, port number	port status change	The port data has been removed.
	lldp_receivedPortDataState	dpid, port number	topology change	The port status has been updated by Link Layer Discovery Protocol [31].

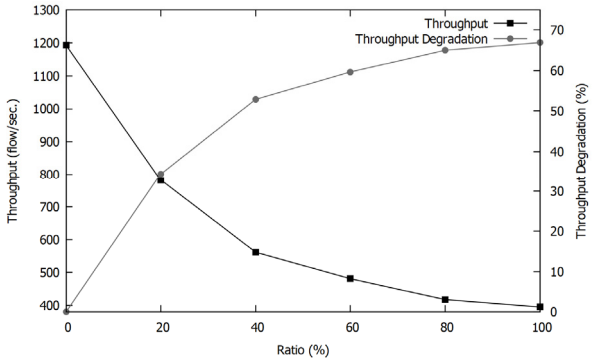
4 Initial Performance Evaluation

For experiment, we setup an SDN environment for emulating the developed system. Since the work proposed in this paper is still a work-in-progress development. Initially, there are two controllers deployed in the test environment for proof-of-concept, one is assigned to Active Controller, and the other one is assigned to Standby Controller. There are three groups of hardware used for experiment: **low-end**, **normal**, and **high-end**. In low-end group, the controller hardware is using Axiomtek NA343R with Intel N3160 CPU to emulate embedded environment; in normal group, the hardware is using PC with Intel G2020 CPU to emulate general environment; in high-end group, we select HP DL360 Gen9 (type 755258-B21) server with Intel E5-2620 CPU to emulate powerful node in data-center environment for evaluation. By doing this, we are able to compare the performance results of different hardware capacities. Furthermore, the controller development is based on RYU (version 3.22) software. The deployed OpenFlow-enabled switch is using Caswell CAR3000 equipped with Intel E5300 CPU, and Intel 82574L network adapter, running OpenvSwitch software (version 2.3.1 on Ubuntu 14.04 operating system) for supporting SDN control. Besides, two end-hosts equipped with ThinkPad x201i (type 3626-NU4) are used to support all the experiments in our evaluation.

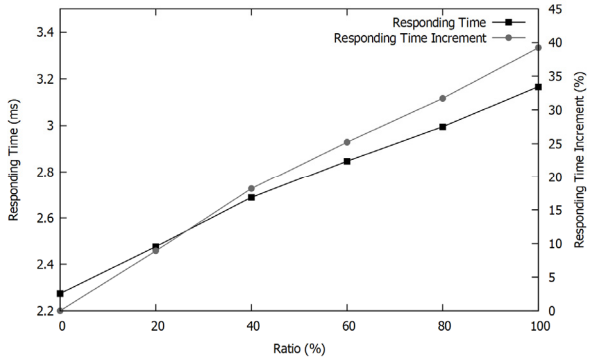
4.1 Throughput

As discussed in the previous section, only those control messages which affect network behavior are recommended to be supervised. Therefore, the first experiment is to evaluate the system performance for processing control messages in limited time. A message generation mechanism is deployed to constantly trigger switch-controller communications in this experiment. When the Active Controller receives the supervised message, it will start the replication process for this message. There is a calculator deployed to monitor the process of control messages. The throughput value is defined as the amount of successfully updated flow rules in the flow table of controller. In experiment, we evaluate the throughput without examination first (i.e., none of the controller messages are supervised). In the next step, we adjust the ratio of supervised messages (i.e., FLOW_ REMOVED) versus non-supervised messages for emulating the throughput employing examination. As the result, the number of processed control messages reduced according to the message ratio. The comparison is shown in Figure 6(a), Figure 6(c) and Figure 6(e). When all messages in the control plane are supervised, it can be found that the degradation of

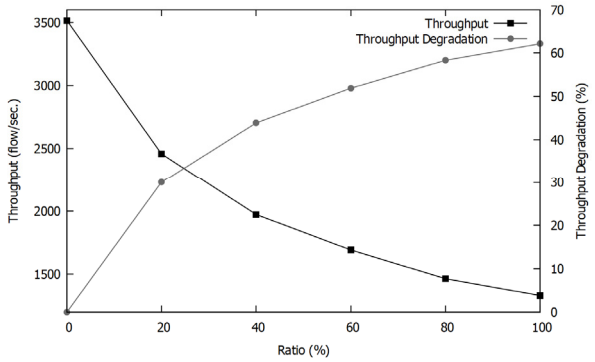
throughput of normal group is about 62 percent in average. In our investigation, it is the message identification actions drags the performance. In our future work, we plan to improve the ICC module for achieving better throughput.



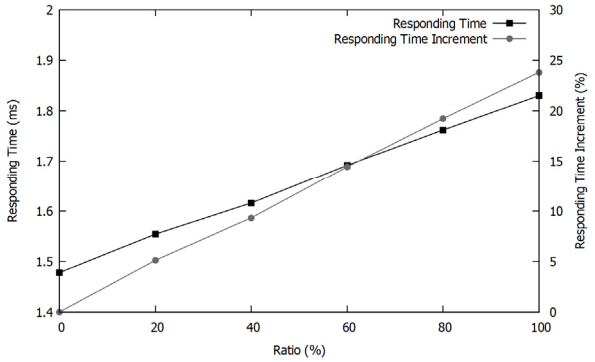
(a) Throughput (low-end hardware group)



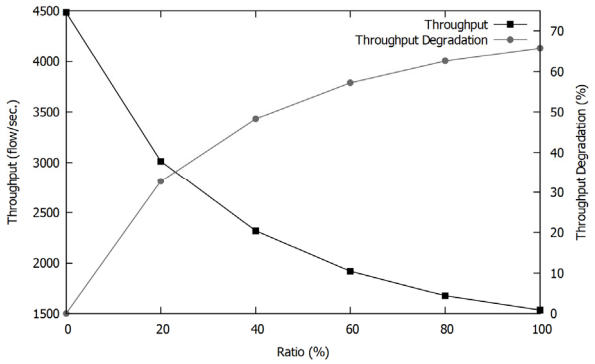
(b) Latency (low-end hardware group)



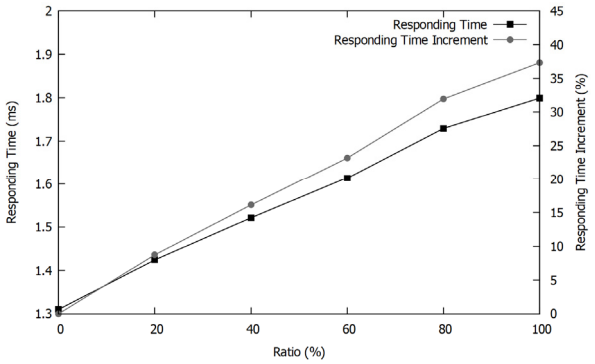
(c) Throughput (normal hardware group)



(d) Latency (normal hardware group)



(e) Throughput (high-end hardware group)



(f) Latency (high-end hardware group)

Figure 6. The testing results of throughput and latency experiments

4.2 Responding Time

The time used for updating the switch is an important index for evaluating SDN controller performance. By observing the responding time, we can realize the latency while waiting for instruction from the switch. We use the same experiment environment as previously. When the Active Controller receives a control message sent by the switch, it starts the examination process for supervised messages. Afterwards, the Active Controller sends a reply message to the switch to complete the controller action. In this experiment, the responding time is measured by the SDN switch. The latency is defined as the time difference between the switch sending and receiving control messages for each new query. The ratio of supervised messages is adjusted in a similar manner as previously. The experiment has been repeated multiple

times, and the measurements for average latency and degradation are shown in Figure 6(b), Figure 6(d), and Figure 6(f).

4.3 Performance Comparison

To optimize the controller operation, we also adjust the heartbeat interval time and batch size in exploration. According to our investigation, it is the state synchronization action makes influence on controller overhead in our development mainly. In throughput increment experiment, we have tried to enlarge the batch size in message examination. Figure 7 shows the experiment results of different supervised message ratio in three hardware groups. Based on our observation, there is always a trade-off between performance and reliability in controller operation. To have the complete control message supervision, the cost will lower down the system throughput. In our opinion, supervising more types of control messages gains more overheads in message examination. Determining which control messages is involved in protection is important to make the balance.

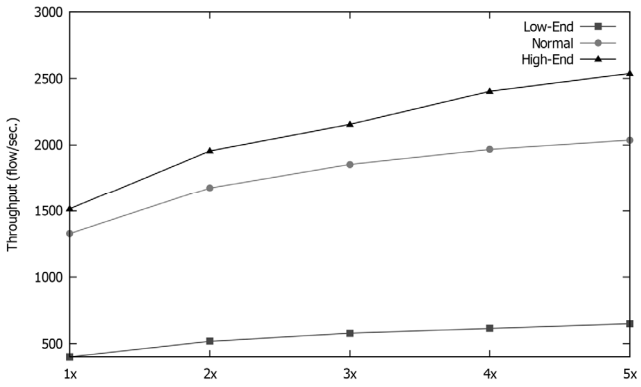


Figure 7. The testing results of batch size adjustment

For appraising the performance in normal network operation, we decided to use benchmark tools for verification. The Cbench [33] is a famous controller benchmark tool of OFLOPS project [34-35] built for OpenFlow component evaluation. The KC-cbench [36] is an extended version of Cbench. We modified its source code to match our testing requirements. We compared our development (normal hardware group) with the classic Ryu controller and another Python-based, single-thread SDN controller (i.e., POX [37]) for comparison with a fair base-line. The results of each measurement (indexed by minimum, maximum, and average) is illustrated in Figure 8. In this figure, the POX controller shows a better throughput with the other two controllers. In our opinion, it is owing to that the POX's framework is lightweight. The controller operation takes less workload in the POX. While the Ryu-based controllers run more functions with event-based control mechanism in operation. For the results of our development and original Ryu controller, the throughput of our development had 60 percent degradation approximately. The reason is that our development requires more process time to handle the control communication for message examination. Compared with other two controllers, our development had lower performance, while it can reduce the chance of switch-and-controller interaction become termination and make the reliability of SDN controller be improved.

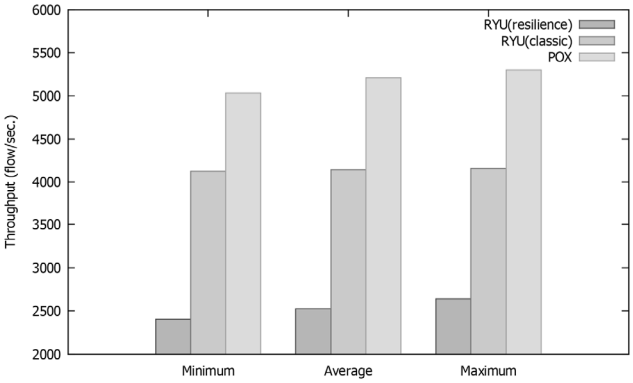


Figure 8. The benchmark results made by KC-cbench

4.4 Use Case Evaluation

For further evaluation, we added more OpenvSwitch to setup an environment for emulating a failover case, Figure 9. In this scenario, Host A operates as a packet generator to keep sending packets to Host B. The packets sent from Host A are forwarded through Switch 1, 2, and 4 originally. For the experiment, we cut off the link between Switch 1 and Switch 2, and also made the Active Controller offline after Switch 1 and Switch 2 sent out the PORT_DOWN STATUS messages. Since the switches were not able to get any instruction from the Active Controller, they were directed to ask a Standby Controller how to forward new arrival packets. At the same time, the modules on the Standby Controller started to take over control responsibility. In this experiment, we measured the time interval between controller failure and recovery. The interruption in the data plane transmission lasted less than two seconds in average. In this use case evaluation, the time interval between controller failure and recovery may be too long for the data plane interruption. After the interpretation, we find out that the reason is probably caused by the controller role change procedure. When the controller system notes the Active Controller is failed, the switches have to start to make the role change, and the Standby Controller turns to be the new Active Controller for all switches. During the changing action, the new arrival packet in data plane may be influenced. While this part is out of the scope of our development. If the switch side is able to speed up the time for controller role change as well as supporting buffering action, the interruption time may be able to reduce.

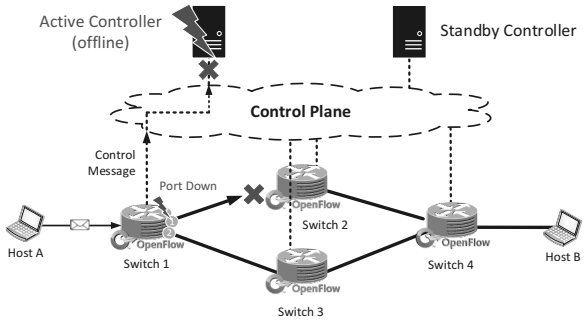


Figure 9. The use case environment built for evaluation

5 Conclusions

To make improvement on the reliability of SDN control plane, this paper introduces a prototype design by using control message replication and examination for enhancing the control resilience of SDN. The developed method is able to be implemented on plural SDN controllers for preventing the behavior lost when controller failure is occurred. According to our evaluation, the throughput of our development is about 60 percent of the original controller, while the degradation makes the network behavior be protected by the replication operation. By doing this, the selected control messages are well examined to make sure the SDN network keeps its constancy during the controller failover.

6 Future Work

In the practical issues, due to the implementation of fault-tolerance in data center often exploits distributed controller like ONOS, the current prototyping system has to be enhanced with for fulfilling such scenarios. When multiple controllers are deployed in the system, implicit the selection of new leader, controllability switching and status synchronization are becoming more complicated, and more practical issues are expected to be investigated. Nevertheless, we are conducting additional evaluation and improvement on system performance, such as adding more Standby Controllers, testing system on datacenter-like networks, and hacking controller framework to speedup reaction time in our SDN testbed [38]. Moreover, we also plan to practice our design on the multi-thread and cluster SDN controller solutions (e.g., NOX-MT [39] and OpenDaylight [40]), trying to verify the operation scenario and evaluating the performance. We will have further exploration and evaluation in our future work for research completeness.

Acknowledgements

The authors would like to thank the anonymous reviewers for their helpful and constructive comments. This work was supported in part by the Ministry of Science and Technology of Taiwan, under contracts No. 104-2221-E-492-002-MY2, 105-2218-E-001-001, and 106-2221-E-006-025. Authors are also grateful to the National Center for High-Performance Computing, TWAREN NOC, and OF@TEIN+ community (Asi@ Connect-17-094) for their help.

References

[1] R. M. Metcalfe, D. R. Boggs, Ethernet: Distributed Packet Switching for Local Computer Networks, *Communications of the ACM*, Vol. 19, No. 7, pp. 395-404, July, 1976.

[2] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, S. Wolf, A Brief History of the Internet, *ACM SIGCOMM Computer Communication Review*, Vol. 39, No. 5, pp. 22-31, October, 2009.

[3] R. Kanagavelu, B. S. Lee, R. F. Miguel, L. Nguyen, L. N. Mingjie, Software Defined Network Based Adaptive Routing for Data Replication in Data Centers, *IEEE International Conference on Networks*, Singapore, 2013, pp. 1-6.

[4] W.-Y. Huang, J.-W. Hu, S.-C. Lin, T.-L. Liu, P.-W. Tsai, C.-S. Yang, F.-I. Yeh, J.-H. Chen, J. Mambretti, Design and Implementation of Automatic Network Topology Discovery System for International Multi-domain Future Internet Testbed, *Journal of Internet Technology*, Vol. 14, No. 2, pp. 181-188, March, 2013.

[5] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, T. Turletti, A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks, *IEEE Communications Surveys & Tutorials*, Vol. 16, No. 3, pp. 1617-1634, February, 2014.

[6] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig, Software-defined Networking: A Comprehensive Survey, *Proceedings of the IEEE*, Vol. 103, No. 1, pp. 14-76, December, 2015.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: Enabling Innovation in Campus Networks, A*CM SIGCOMM Computer Communication Review*, Vol. 38, No. 2, pp. 69-74, April, 2008.

[8] M. A. Fera, S. M. Shalinie, A Survey on Foundation for Future Generation Internet through Network Virtualization, *Proceedings of the International Conference on Advanced Computing*, Chennai, India, 2014, pp. 172-177.

[9] F. J. Ros, P. M. Ruiz, Five Nines of Southbound Reliability in Software-defined Networks, *Proceedings of the Workshop on Hot Topics in Software Defined Networking*, Chicago, IL, 2014, pp. 31-36.

[10] A. Xie, X. Wang, W. Wang, S. Lu, Designing a Disaster-resilient Network with Software Defined Networking, *Proceedings of the IEEE International Symposium of Quality of Service*, Hong Kong, China, 2014, pp. 135-140.

[11] F.-H. Tseng, K.-D. Chang, S.-C. Liao, H.-C. Chao, V. C. M. Leung, sPing: A User-centred Debugging Mechanism for Software Defined Networks, *IET Networks*, Vol. 6, No. 2, pp. 39-46, March, 2017.

[12] A. Lara, A. Kolasani, B. Ramamurthy, Network Innovation Using OpenFlow: A Survey, *Communications Surveys & Tutorials, IEEE*, Vol. 16, No. 1, pp. 493-512, First Quarter, 2014.

[13] M.-Y. Luo, J. Chen, J. Mambretti, S.-W. Lin, F. Yeh, P.-W. Tsai, C.-S. Yang, Network Virtualization Implementation over Global Research Production Networks, *Journal of Internet Technology*, Vol. 14, No. 7, pp. 1061-1072, December, 2013.

[14] Software-Defined Networking (SDN) Definition, 2017, https://www.opennetworking.org/sdn-definition/.

[15] D. Kreutz, F. M. V. Ramos, P. Verissimo, Towards Secure and Dependable Software-defined Networks, *Proceedings of the Workshop on Hot Topics in Software Defined Networking*, Hong Kong, China, 2013, pp. 55-60.

[16] J. P. G. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, P. Smith, Resilience and Survivability in Communication Networks: Strategies, Principles, and Survey of Disciplines, *Computer Networks*, Vol. 54, No. 8, pp. 1245-1265, June, 2010.

[17] A. S. da Silva, P. Smith, A. Mauthe, A. Schaeffer-Filho, Resilience Support in Software-defined Networking: A Survey, *Computer Networks*, Vol. 92, No. 1, pp. 189-207, December, 2015.

[18] Q. Yan, F. R. Yu, Q. Gong, J. Li, Software-defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges, *IEEE Communications Surveys & Tutorials*, Vol. 18, No. 1, pp. 602-622, First Quarter, 2016.

[19] J. Li, J. Hyun, J.-H. Yoo, S. Baik, J. W. K. Hong, Scalable Failover Method for Data Center Networks Using OpenFlow, *Proceedings of the IEEE Network Operations and Management Symposium*, Krakow, Poland, 2014, pp. 1-6.

[20] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, G. Parulkar, ONOS: Towards an Open, Distributed SDN OS, *Proceedings of the Workshop on Hot Topics in Software Defined Networking*, Chicago, IL, 2014, pp. 1-6.

[21] P. Fonseca, R. Bennesby, E. Mota, A. Passito, A Replication Component for Resilient OpenFlow-based Networking, *Proceedings of the IEEE Network Operations and Management Symposium*, Maui, HI, 2012, pp. 933-939.

[22] N. Katta, H. Zhang, M. Freedman, J. Rexford, Ravana: Controller Fault-tolerance in Software-defined Networking, *Proceedings of the ACM SIGCOMM Symposium on Software Defined Networking Research*, Santa Clara, CA, 2015, pp. 1-12.

[23] T. Watanabe, T. Omizo, T. Akiyama, K. Iida, ResilientFlow: Deployments of Distributed Control Channel Maintenance Modules to Recover SDN from Unexpected Failures, *IEICE Transactions on Communications*, Vol. 99, No. 5, pp. 1041-1053, May, 2016.

[24] K. Kuroki, N. Matsumoto, M. Hayashi, Scalable OpenFlow Controller Redundancy Tackling Local and Global Recoveries, *Proceedings of the International Conference on Advances in Future Internet*, Barcelona, Spain, 2013, pp. 61-66.

[25] B. Han, V. Gopalakrishnan, L. Ji, S. Lee, Network Function Virtualization: Challenges and Opportunities for Innovations, *IEEE Communications Magazine*, Vol. 53, No. 2, pp. 90-97, February, 2015.

[26] A. Nakao, Software-defined Data Plane Enhancing SDN and NFV, *IEICE Transactions on Communications*, Vol. 98, No. 1, pp. 12-19, January, 2015.

[27] P. Fonseca, R. Bennesby, E. Mota, A. Passito, Resilience of SDNs Based on Active and Passive Replication Mechanisms, *Proceedings of the IEEE Global Communications Conference*, Atlanta, GA, 2013, pp. 2188-2193.

[28] Ryu, https://osrg.github.io/ryu/.

[29] Open vSwitch, http://openvswitch.org/.

[30] R. Guerraoui, L. Rodrigues, *Introduction to Distributed Algorithms*, Springer, 2004.

[31] P. Congdon, B. Lane, *IEEE Standard 802.1AB - Station and Media Access Control Connectivity Discovery*, http://www.ieee802. org/1/pages/802.1ab.html.

[32] OpenFlow specification version 1.3.0, wire protocol 0x04, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf.

[33] Cbench, https://github.com/mininet/oflops/tree/master/cbench.

[34] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, A. W. Moore, OFLOPS: An Open Framework for OpenFlow Switch Evaluation, *Proceedings of the International Conference on Passive and Active Network Measurement*, Vienna, Austria, 2012, pp. 85-95.

[35] OFLOPS: OpenFlow Operations Per Second, https://github.com/andi-bigswitch/oflops.

[36] KulCloud cbench, https://kulcloud.wordpress.com/tag/cbench.

[37] POX, https://github.com/noxrepo/pox.

[38] P.-W. Tsai, P.-W. Cheng, H.-Y. Chou, M.-Y. Luo, C.-S. Yang, Toward Inter-Connection on OpenFlow Research Networks, *Proceedings of the 36th Asia-Pacific Advanced Network*, Daejeon, Korea, 2013, pp. 9-16.

[39] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, R. Sherwood, On Controller Performance in Software-defined Networks, *Proceedings of the USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, San Jose, CA, 2012, pp. 1-6.

[40] OpenDaylight, https://www.opendaylight.org/.

Biographies

Pang-Wei Tsai received the B.S. degree in Electrical Engineering, and the M.S. as well as Ph.D. degrees in Computer and Communication Engineering from National Cheng Kung University. His research interests include software-defined networking, cloud computing, network management, CPS security, and network testbed. Currently, he is working in Delta Research Center, Delta Electronics, Inc.

Wai-Hong Fong received the B.S. degree in Electrical Engineering and the M.S. degree in Computer and Communication Engineering from National Cheng Kung University. He studied OpenFlow network, fault tolerance and embedded system during his academic career, and now works for Synology, Inc.

Wu-Hsien Chang received the B.S. degree in Computer Science and Information Engineering from National Chung Cheng University, and the M.S. degree in Computer and Communication Engineering from National Cheng Kung University. His research interests include software-defined networking and network testbed, and he is currently working in Telecommunication Laboratories, Chunghwa Telecom Co., Ltd.

Chu-Sing Yang is a Professor of Electrical Engineering in the Institute of Computer and Communication Engineering at National Cheng Kung University (NCKU). He joined the faculty of the Department of Electrical Engineering at National Sun Yat-Sen University (NSYSU) in 1988. He was the chair of the Department of Computer Science and Engineering from August 1995 to July 1999, and the director of the Computer Center from 1998 to 2002. He joined the faculty of the Department of Electrical Engineering at NCKU in 2006. His research interests include software-defined networking, network management, cloud computing, and cyber-security.