Future Generation Computer Systems 95 (2019) 681–693

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Dynamic slave controller assignment for enhancing control plane robustness in software-defined networks

Tao Hu [a], Peng Yi [a,*], Zehua Guo [b,*], Julong Lan [a], Yuxiang Hu [a]

[a] National Digital Switching System Engineering and Technological Research Center, Zhengzhou 450002, China
[b] University of Minnesota Twin Cities, Minneapolis, MN 55455, USA

HIGHLIGHTS

- This paper proposes a dynamic slave controller assignment that prevents the network crash by planning slave controller assignment ahead of the controller failures.
- We identify the controller chain failure phenomenon caused by existing slave assignment schemes and illustrate it with examples.
- We formulate the SCA problem that considers the latency between the switch and controller, the load balancing among controllers after the controller failures, the robustness of controller, and prove its NP-completeness.
- We design DSCA to solve SCA problem.

ARTICLE INFO

Article history:
Received 25 September 2018
Received in revised form 4 November 2018
Accepted 7 January 2019
Available online 1 February 2019

Keywords:
Software-defined networking
Control plane
Multi-controller
Controller failure
Fault-tolerance

ABSTRACT

Multi-controller is a scalable control plane solution for the large-scale Software-Defined Networking (SDN). To achieve high resilience, an SDN switch can connect one master controller for normal operation and one slave controller that backup the function of the master controller. Once the master controller fails, one of the slave controllers will be assigned to switches to works as the new master controller. However, the inappropriate slave controller assignment may cause controller chain failure, where running out of the capacity of the assigned controller, even crash the entire network. In this paper, we propose a dynamic slave controller assignment that prevents the network crash by planning slave controller assignment ahead of the controller failures. We first describe the controller chain failure phenomenon: due to unreasonable slave controller assignment, the entire network may crash when one controller fails. To prevent the phenomenon, we formulate the slave controller assignment problem as a multi-objective mixed optimization problem that considers latency, load balancing and robustness, and prove its NP-complete complexity. We solve the problem with a dynamic slave controller assignment (DSCA) scheme. It firstly checks whether there are controller failures in state detection module, then completes the elastic slave assignment and generates a new slave assignment for switches in efficient slave assignment module. Finally, in role adjustment module, it changes the roles of some controllers and reconnects switches. Simulation results show our solution can decrease the worst case latency under controller failures by 35.1% averagely, and reduce the probability of network crash.

1. Introduction

Software-Defined Networking (SDN) innovates the design philosophy of network with the centralized control, open interface and network programmability [1]. Many networks (e.g., data center (DC) [2], wide area network (WAN) [3]) begin to employ SDN to improve network performance and service quality. The original SDN design only deploys one controller in the network, and it does not have enough control abilities to manage the network with increasing traffic and scale [4]. Several researchers propose to deploy the logically centralized but physically distributed multi-controller to improve the scalability of the control plane for multi-domain SDN, such as HyperFlow [5], and Kandoo [6]. In the multi-domain network, each domain has one local controller named the domain controller to manage the switches and process flow requests from its domain, and controllers of different domains communicate with each other about their domain information to ensure a consistent network view [7].

As the number of underlying network devices and traffic increase, the controller will have a high burden on processing flow requests, leading to a higher probability of controller failure. A

* Corresponding authors.
E-mail addresses: yipengndsc@163.com (P. Yi), guolizihao@hotmail.com (Z. Guo).

https://doi.org/10.1016/j.future.2019.01.010
0167-739X/

controller could also fail when the server or virtual machine running the controller experiences the hardware failure or software crash [8]. In order to improve the resiliency of the multi-controller control plane, some SDN protocols (e.g., OpenFlow 1.2) propose to use a backup mechanism [9]. Each controller has two roles: master and slave. Typically, for a switch, its master controller is responsible for processing the flow requests, while its slave controller is used for backup. Each controller informs its role to switches via a role request message. In a domain, the domain controller plays a master role for switches in the domain, but the switches can also connect to the controllers of other domains and set them as the slave roles. If the master controller of one switch does not work, this switch will be reconnected to its predetermined slave controller, which will work as a new master controller for the switch.

However, few researchers consider how to choose the slave controllers for switches, especially under the controller failure scene. Inefficient slave controller assignment (e.g., statically assigning slave controller to switch according to the hop relationship) could seriously degrade network performance and even lead to controller chain failure, which would collapse the overall network. Moreover, the controller chain failure can be described as followings: one controller failure can cause another controller failure, and even crash the entire network in the worst case. More details will be shown in Section 2.

Motivated by these concerns, in this paper, we propose a dynamic slave controller assignment (DSCA) scheme to construct a fault-tolerant control plane. We formulate a slave controller assignment (SCA) problem, considering latency, load balancing and robustness to assign the slave controller for switches. DSCA not only considers the availability and capacity of controllers but also plans ahead for controller failures to prevent network crashing. In DSCA, we consider that one or more controllers could fail, and our primary objective is to dynamically assign slave controllers to switches' new masters to guarantee the good fault-tolerant performance of control plane under controller failures. We summarize our main contributions as follows:

- We identify the controller chain failure phenomenon caused by existing slave assignment schemes and illustrate it with examples.
- We formulate the SCA problem that considers the latency between the switch and controller, the load balancing among controllers after the controller failures, the robustness of controller, and prove its NP-completeness.
- We design DSCA to solve the SCA problem by collaborating the designed modules. Once identifying controller failures in state detection module, efficient slave assignment module will implement the elastic slave assignment, and then role adjustment module changes the roles of some controllers. Besides, the improved heuristic algorithms are loaded on the corresponding modules to increase efficiency.
- We evaluate the performance of DSCA against baseline schemes. Results show that DSCA can effectively reduce the worst case latency under controller failures by 35.1% averagely, and decrease the probability of network crashing.

The rest of paper is organized as follows. Section 2 presents the motivation of this paper. Section 3 models and formulates SCA problem in the network. Section 4 presents the DSCA. Section 5 exhibits our performance evaluation. Section 6 introduces the related works. Section 7 concludes the paper.

2. Motivation

In this section, we firstly introduce the controller chain failure existed in the existing slave controller assignment, and then explain why the slave controller assignment needs to be improved.

2.1. Controller chain failure

In the distributed multi-domain and multi-controller SDN network, when a master controller fails due to hardware or software failure, the predetermined slave controllers will be instantly assigned to the switches managed by the failed controller. Although both the multi-controller and backup mechanism (e.g., master and slave) have improved the scalability and reliability of SDN network, the existing works have no sufficient research on planning slave controller assignment after controller failure. Generally, the existing slave controller assignment rigidly predetermines the nearest controller of switch as slave role, but this assignment is flawed and will bring about a potential threat introduced as controller chain failure. Further, we will use a simple example in Fig. 1 to illustrate controller chain failure phenomenon.

Fig. 1(a) shows an SDN network with three controllers (C1–C3) and six switches (S1–S6). The network is partitioned into three domains (Domains 1–3). One controller can control at most three switches. In case of controller C1 failure, the switches in the Domain 1 are disconnected with C1, as shown in Fig. 1(b). According to the existing slave controller assignment, the nearest controller C2 is assigned to the switches of Domain 1, and its role is changed from slave to master. Thus, all switches of the failed controller C1 are connected to C2. Unfortunately, due to the constraint of controller capacity, C2 cannot simultaneously manage its own domain's switches (S3 and S4) and the newly added switch (S1 and S2). The flow requests sent by these four switches (S1 to S4) will exhaust the capacity of C2, then C2 fails, as shown in Fig. 1(c). Similarly, in Fig. 1(d), S1 to S4 will be reconnected to C3, and C3 also fails because of running out of controller capacity. Finally, all controllers in the network are in the failed state, and the entire network is crashed.

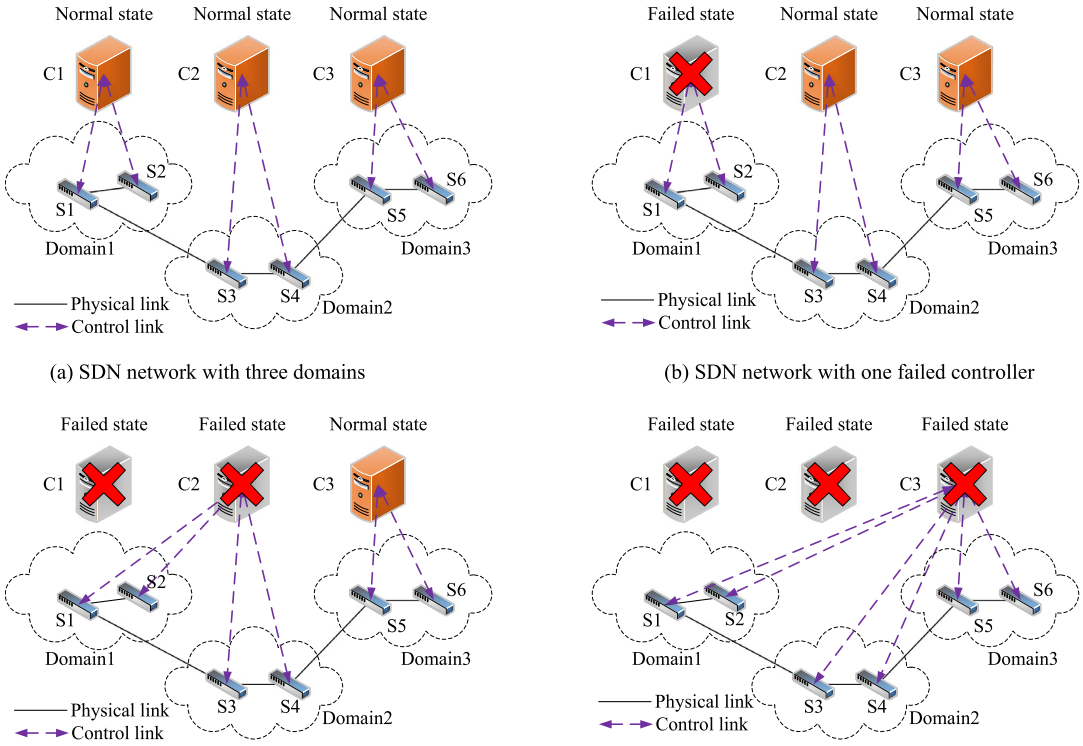
Therefore, it presents an interesting phenomenon by this example: in multi-controller SDN network, one controller failure can cause another controller failure, and even crash the entire network in the worst case. We call this phenomenon *controller chain failure*.

2.2. Requirement for improving slave controller assignment

Based on the above example, we can find that the existing slave controller assignment is defective and likely to generate controller chain failure that causes serious damage to the network. Therefore, it is necessary to improve slave controller assignment to achieve the fault-tolerant control plane. Through analyzing, the primary cause of this problem is the static and inflexible assignment of slave controllers without considering the overall situation of network (e.g., controller capacity, flow request). Thus, we propose a new idea to improve the slave controller assignment, which not only considers the capacity and availability of controller, but also dynamically determines switches' slave controllers based on network condition.

For example, in Fig. 2, when C1 fails in the network, both S1 and S2 need to connect to new controller. Differing from the existing scheme, we firstly determine C2 and C3 as the slave controller of S1 and S2, respectively. Then, C2 and C3 will be assigned to S1 and S2 to act as the switches' master controllers. Obviously, this improvement has two advantages: (1) controller capacity is fully utilized but not exceeds the maximum threshold; (2) the controller chain failure has been eliminated effectively.

Therefore, integrated with the above examples and analyses, we can conclude that the improvement for slave controller assignment is necessary, and our proposal could minimize the impact of controller failure on network and further enhance the fault-tolerant ability of control plane.



(a) SDN network with three domains

(b) SDN network with one failed controller

(c) SDN network with two failed controllers

(d) SDN network with three failed controllers

Fig. 1. An example of controller chain failure.

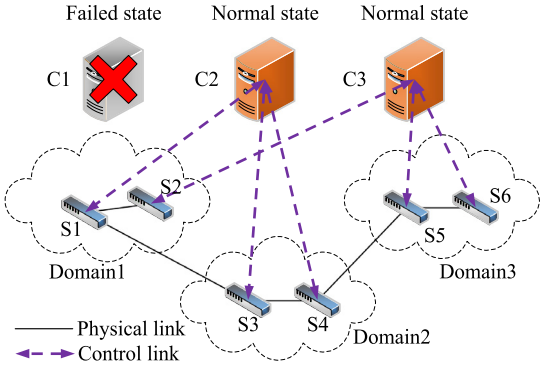


Fig. 2. Our proposal for improving slave controller assignment.

Table 1
Notations.

Notation	Definition
$V = \{v_i\}$	Set of nodes
$E = \{e_{ij}\}$	Set of links
M	Number of controllers
N	Number of switches
$C = \{c_m\}$	Set of controllers
FC	Set of failed controllers
$S = \{s_n\}$	Set of switches
s_n^m	Binary variable: $s_n^m = 1$ shows c_m is master of s_n, otherwise $s_n^m = 0$
$s_n(k)$	Binary variable: $s_n(k) = 1$ shows c_k is slave of s_n, otherwise $s_n(k) = 0$
$D = \{D_m\}$	Set of domains
$f(t)_n$	Flow request of switch s_n in the time t
U_m	Capacity of controller c_m
d_{ij}	Latency of shortest path between node v_i and v_j.
L_m	Load of controller c_m

3. Modeling and formulation

In this section, we give the detailed modeling of slave controller assignment in the multi-controller SDN network, which mainly considers latency, load balancing and robustness. Then, we formulate the slave controller assignment (SCA) problem as the multi-objective mixed optimization and prove its NP-completeness.

3.1. Network model

SDN network is represented by an undirected graph $G = (V, E)$, where V and E are node set and link set, respectively. $C = \{c_1, c_2, \ldots, c_m\}$ and $S = \{s_1, s_2, \ldots, s_n\}$ are the sets of controllers and switches deployed in the network. We assume all controllers have been deployed in the topology in advance [10]. Meanwhile, the entire network is partitioned into several domains, and each domain only deploys one controller. According to OpenFlow protocol, a controller can not only act as the master role for one switch, but also as the slave role for another switch. Here, in order to simplify the connection relationships, we assume that each switch can be simultaneously connected to one controller as its master and another one as its slave. Specifically, we define binary variable s_n^m, where $s_n^m = 1$ represents c_m is the master controller of switch s_n, otherwise $s_n^m = 0$. Meanwhile, binary variable $s_n(k) = 1$ represents c_k is the slave controller of switch s_n, otherwise $s_n(k) = 0$. Please note that the flow request of a switch refers to the number of flows whose header does not match with any of the flow table entries, and all unmatched flows are forwarded to its master controller. Correspondingly, the controller's loads are the sum of flow requests sent by all switches in the same domain. The primary notations are summarized in Table 1.

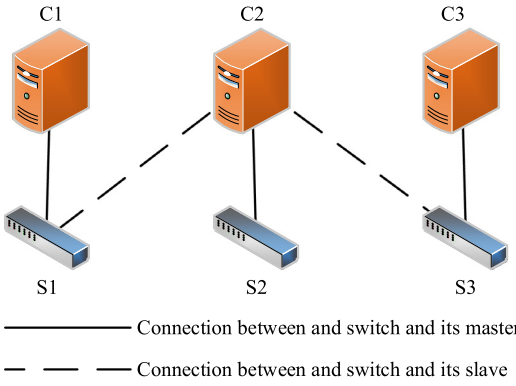


Fig. 3. The connection relationships between switch and controller.

3.2. Slave controller assignment formulation

The objective of this paper is to achieve a fault-tolerant control plane through improving slave controller assignment under the controller failure scene. Therefore, we firstly introduce some relative metrics that include latency, load balancing and robustness, to evaluate the slave controller assignment. Then, we formally formulate the slave controller assignment problem.

3.2.1. Latency between switch and controller

Latency is an important evaluation index in SDN network, and here we mainly consider the latency between switches and controllers and relax the latency between controllers. Therefore, the latency between switch s_n and controller c_m is d_{mn}, which represents the latency of the shortest path. The existing researches mainly focus on optimizing the latency between switch and its master controller, but there is a lack of planning for the latency between switch and its slave controller. Once the switch of the failed controller connects to its slave controller that has the large latency, the rate of flow transmission will be degraded obviously. For switch s_n and its slave controller c_k, we compute slave controller latency D^{slave}_{n-k} in Eq. (1). Specifically, one controller can be simultaneously assigned to several switches as slave. For example, in Fig. 3, C2 plays as a slave controller for S1 and S3. Therefore, D^{slave}_{k} is the sum of latencies of controller c_k acting as slave role in Eq. (2), and D^{slave} is the sum of latencies of all controllers acting as slave roles in Eq. (3).

$$D^{slave}_{n-k} = d_{nk} \cdot s_n(k) \tag{1}$$

$$D^{slave}_{k} = \sum_{n=1}^{N} d_{nk} \cdot s_n(k) \tag{2}$$

$$D^{slave} = \sum_{k=1}^{M} \sum_{n=1}^{N} d_{nk} \cdot s_n(k) \tag{3}$$

3.2.2. Controller load balancing

The purpose of load balancing is to ensure that each controller capacity is effectively allocated and equitably used, thereby satisfying the network's QoS requirements. In general, the load of controller mainly focuses on processing the flow requests (e.g., Packet-in messages) sent by switches. Therefore, in order to facilitate the subsequent calculation, we define the controller load and load variance.

Definition 1 (*Controller Loads*)**.** In an SDN network, controllers are responsible for processing the flow requests of switches based on the global view. Therefore, the flow request rates of switches in one domain are defined as the loads of domain controller. The loads of controller c_m could be computed with Eq. (4).

$$L_m = \sum_{n=1}^{N} s^m_n \cdot f\ (t)_n \tag{4}$$

Definition 2 (*Load Variance*)**.** Based on Definition 1, we can get the loads of each controller. To express the controller load balancing, we introduce load variance computed in Eq. (5), where $\bar{L}$ is the average value of controller loads. According to the mathematical meaning of the variance, we can conclude that the smaller *LV*, the more balanced the distribution of controller loads.

$$LV = \frac{1}{M} \sum_{m=1}^{M} (L_m - \bar{L})^2 \tag{5}$$

$$\bar{L} = \frac{1}{M} \sum_{m=1}^{M} L_m \tag{6}$$

Because the focus of this paper is improving the fault-tolerant ability of control plane, we assume controller loads condition of normal network has been optimized in advance by referring [11]. When there are controller failures in the network, we consider the load balancing after the slave controller has been assigned to switches of the failed controller as the master roles. Thus, the reassigned controller loads consist of two parts: the original loads and the newly added loads, as shown in Eq. (7). We suppose there are r failed controllers in the network. Then, we recalculate the load variance $LV_{failure}$ after controller failures with Eq. (8).

$$L^*_k = \sum_{n=1}^{N} s^k_n \cdot f\ (t)_n + \sum_{i=1}^{N} s_i(k) \cdot f\ (t)_i \tag{7}$$

$$LV_{failure} = \frac{1}{M-r} \sum_{k=1}^{M-r} (L^*_k - \frac{1}{M-r} \sum_{k=1}^{M} L^*_k)^2 \tag{8}$$

3.2.3. Robustness of slave controller assignment

In the large-scale SDN, the flow request rate of switch shows strong dynamism because of complex traffic requirements [12]. The dynamic changes in demand result in dynamic changes of the utilization of capacity for controller. Assigning the undercapacity controller to the switch, which has the high flow request rate, may bring about new controller failure because of running out of controller capacity. Therefore, the robustness is also an important consideration when assigning slave controllers to switches.

There are two assignment patterns from the perspective of network traffic. On the one hand, assigning slave controller to switch is based on peak flow request rate. This way can effectively avoid controller chain failure once the master controller fails. But it requires the slave controller to reserve plenty of idle capacities, lowing the utilization of controller capacity. On the other hand, assigning slave controller to switch is based on the average flow request rate. This method can reduce the reserved capacity of slave controller but not adapt to the network with drastic fluctuating traffic. Thus, it is necessary to fully consider the flow request of switch to assign slave controller, ensuring the robustness.

In terms of the dynamic flow request rate of switch, its expectation and standard deviation can be computed based on historical log data. For a given switch, the expectation of flow request rate represents the average level of flow consumed controller capacity. Moreover, the standard deviation of flow request rate reflects how much the flow fluctuates. Therefore, through referring the expectation and standard deviation of switch's flow request rate, we could achieve better robustness of slave controller assignment if the appropriate controller with rational capacity is assigned to the switch as the slave.

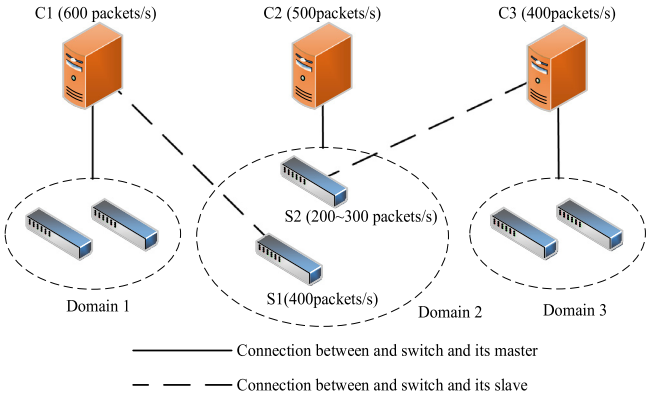


Fig. 4. An example of robustness of slave controller assignment.

For example, in Fig. 4, there are three SDN domains, and the available capacities of three controllers are 600 packets/s (C1), 500 packets/s (C2) and 400 packets/s (C3), respectively. In domain 2, switch S1's flow request rate is almost steady (400 packets/s), while S2's flow request rate is fluctuated (200∼300 packets/s). Considering the robustness, when C1 and C3 are assigned to S1 and S2 as slave controller respectively, the performance of the control plane can be guaranteed even if C2 fails. It is also helpful to address controller chain failure (Section 2).

Based on the idea mentioned above, we introduce some metrics to describe the robustness of slave controller assignment. $E(f(t)_n)$ and $\sigma(f(t)_n)$ are denoted as the expectation value and the standard deviations of flow request rate of switch s_n. Then, we analyze the usage of the controller capacity in detail.

Definition 3 (*Control Capacity of a Controller*)**.** One controller deployed in a SDN domain should control the switches in the domain, and those switches will occupy the corresponding capacity, which is defined as control capacity. Thus, the control capacity of controller c_k is computed as U_k^{Con} in Eq. (9), where s_n^k shows c_k is the master controller of s_n.

$$U_k^{Con} = \sum_{n=1}^{N} s_n^k \cdot f\ (t)_n \tag{9}$$

Definition 4 (*Reserved Capacity of Controller*)**.** For a controller, when it is assigned to some switches as slave role, it must reserve ahead certain capacity for slave connections. Therefore, we call the reserved capacity of controller c_k as U_k^{Res} in Eq. (10), where $s_i(k)$ shows c_k is the slave controller of s_i.

$$U_k^{Res} = \sum_{i=1}^{N} E(f(t)_i) \cdot s_i(k) \tag{10}$$

When a controller is ready to play the slave role for some switches, the sum of its control capacity and its reserved capacity must not exceed the maximum threshold of capacity, as shown in Eq. (11).

$$U_k^{Con} + U_k^{Res} \leq U_k \tag{11}$$

Based on the above definitions, when evaluating the robustness of slave controller assignment, we must consider maximum capacity, control capacity and reserved capacity of the controller. Meanwhile, due to the fluctuant flow request rate of switch, we introduce an assignment weight, which is related to the standard deviation of flow request rate, to correct the result. For example, the assignment weight w_k of slave controller c_k is computed in Eq. (12). Further, the robustness of slave controller c_k can be expressed in Eq. (13).

$$w_k = \frac{\sum_{n=1}^{N} \sigma(f\ (t)_n) \cdot s_n(k)}{\sum_{n=1}^{N} \sigma(f\ (t)_n)} \tag{12}$$

$$\begin{aligned} R_k &= (U_k - U_k^{Con} - U_k^{Res}) \cdot w_k \\ &= (U_k - U_k^{Con} - U_k^{Res}) \cdot \frac{\sum_{n=1}^{N} \sigma(f\ (t)_n) \cdot s_n(k)}{\sum_{n=1}^{N} \sigma(f\ (t)_n)} \end{aligned} \tag{13}$$

The average of the robustness of all slave controllers is defined as R in Eq. (14), where M' is the number of controllers acted as slaves. Generally, all controllers in the network should participate in the slave assignment to ensure the full utilization of control resources.

$$R = \frac{1}{M'} \sum_{k=1}^{M'} R_k \tag{14}$$

3.2.4. The SCA problem formulation

The purpose of this paper is to achieve the fault-tolerant control plane through improving slave controller assignment. Therefore, how to reasonably assign controllers as slaves to switches becomes a key point, and we define it as *slave controller assignment* (SCA) *problem*. Based on the above metrics (latency, load balancing, robustness), we formulate SCA as follows.

$$\text{Objective} \quad \min F = \{F_1, F_2, F_3\} \tag{15}$$

Subjected to

$$\begin{cases} F_1 = D^{slave} \\ F_1 = \left|LV - LV_{failure}\right| \\ F_3 = Z - R \end{cases} \tag{16}$$

$$\forall n \in N \quad \sum_{m=1}^{M} s_n^m = 1 \tag{17}$$

$$\forall n \in N \quad \sum_{m=1}^{M} s_n(k) = 1 \tag{18}$$

$$0 \leq |FC| \leq M \tag{19}$$

$$|FC| \leq |C - FC| \tag{20}$$

$$\forall c_m \in C, \sum_{n=1}^{N} s_n^m \cdot f(t)_n < U_m \tag{21}$$

$$\sum_{m=1}^{M} \sum_{n=1}^{N} s_n^m \cdot f(t)_n < \sum_{m=1}^{M} U_m \tag{22}$$

$$\forall n \in N, m \in M \quad s_n^m \in \{0, 1\} \tag{23}$$

$$\forall n \in N, m \in M \quad s_n(k) \in \{0, 1\} \tag{24}$$

Eq. (15) shows the objective function of the multi-criteria optimization problem. Eq. (16) represents the multiple criteria, including latency, controller load balancing, and robustness, where LV and $LV_{failure}$ respectively indicate the load variance of controllers in normal state and failed state, and Z is a constant. To normalize the model, the robustness is subtracted by a fixed constant to transfer to minimize. Eq. (17) shows a switch only has one master controller in the network. Eq. (18) indicates each switch can select one controller as slave role from its backup controller set. Eq. (19) promises there is no chance of all controllers failing in the network. Eq. (20) presents failed controllers' quantity is less than the normal controllers'. Eqs. (21) and (22) show that controllers have enough capacities to process the flow requests sent from their controlled switches. Eqs. (23) and (24) are integral constraints which ensure that all the decision variables take values either 0 or 1.

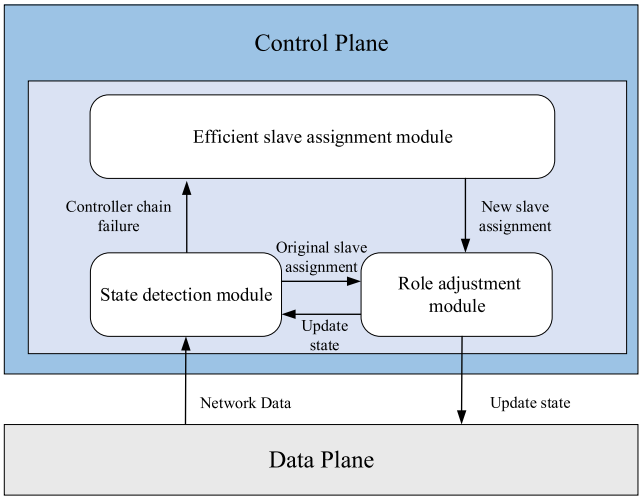


Fig. 5. DSCA architecture for fault-tolerant control plane.

Theorem 1. *SCA is NP complete.*

Proof. We prove the NP completeness of SCA by considering a decision version of the problem, and showing a reduction from controller placement problem (CPP) [10]. Moreover, CPP has proved the NP completeness of master controller assignment. An instance of CPP is: for a given SDN network G with a controller set C and a switch set S, is there a subset $C' \in C$ such that $\sum_{C' \in C} s_n^c \neq 0$? For a comparison, we set an instance of SCA. We assume that there are two controllers c_1, c_2 and $|S|$ switches. When assuming c_1 as master controller, it meets CPP and $\sum_{C' \in C} s_n^{c_1} \neq 0$. Then, we can get $c_2 \in C \backslash \{c_1\}$. At this time, SCA still assigns c_2 to one of switch from S as slave. Thus, this operation can be reduced to a specific CPP with one determined controller. Similarly, $C' \in C \backslash \{c_1\}$ and $\sum_{c_2 \in C} s_n^{c_2} \neq 0$. Finally, the reductions can be done within the polynomial time, which completes the proof.

4. Dynamic slave controller assignment

In this section, we introduce a dynamic slave controller assignment (DSCA) scheme, which includes the modules of state detection, efficient slave assignment, and role adjustment, to solve the SCA problem formulated in Section 3.

4.1. Overview of DSCA

Fig. 5 shows the overview of DSCA, which consists of three parts: state detection module, efficient slave assignment module, and role adjustment module. The input of DSCA is the network data, including the topology connection and flow request rates of switches.

State detection module, as the first module, checks whether there are controller failures. Further, it has two output results: if the existing failed controllers cause controller chain failure (described in Section 2), they will be labeled and delivered to efficient slave assignment module. Otherwise, they are delivered to role adjustment module to implement original slave assignment.

Efficient slave assignment module is the core part of DSCA, and it will complete the elastic slave assignment according to the controller failure condition and generate a new slave assignment for switches.

Role adjustment module changes the roles (e.g. master and slave) of some controllers based on the types of slave assignment (new or original). Finally, the adjustment results are respectively delivered to state detection module and data plane to update network states, including the load information of controllers and connection relationships between switches and controllers.

Based on three modules design and the corresponding feedback process, DSCA can effectively weak the impact of controller failure on the network and construct the fault-tolerant control plane. The detailed introductions are described as follows.

4.2. State detection module

In DSCA, state detection module is mainly responsible for detecting controller failures and further determines the type of failures. Specifically, Fig. 6 shows the workflow of the module. Firstly, this module will collect the network data (e.g. topology connection and flow request rates of switches) from data plane. We assume that the control plane has generated an original slave assignment scheme in the stage of network design to cope with uncertain controller failure, such as assigning the nearest controller to switch as slave. Then, it will judge whether there are controller failures in the network. Here, the controller failure includes two meanings: hardware failure due to sudden breakdown, and software failure caused by the depletion of controller capacity. Both failures will make controller lose efficiency. In the multi-controller SDN network, controllers can periodically synchronize state information with each other to maintain the network status consistency. In this paper, we follow this idea and introduce a simple and effective controller failure checking method by referring to the existing works [13–15]. Once several controllers cannot receive the heartbeat messages that are sent from the specific controller during a state synchronization, then we consider this controller is failed. For example, a network has three controllers: C1, C2, and C3. If controllers C1 and C2 cannot simultaneously synchronize network state with C3 through heartbeat messages, we consider C3 is failed. Further, for these failed controllers, this module judges whether they will cause controller chain failure by precomputing the original slave assignment. If so, this kind of controllers will be added into the failed controller set FC and then go to efficient slave assignment module. Otherwise, the failed controllers will go to the role adjustment module to implement the original slave assignment.

4.3. Efficient slave assignment module

Efficient slave assignment module is the key part of DSCA, and it can implement the rational slave controller assignment to the switches of failed controllers to avoid controller chain failure and achieve fault tolerance. In Section 3, we have formulated the SCA problem and proved its NP-completeness. SCA problem has three optimization objectives, which include minimizing latency, minimizing load variance and maximizing the robustness. For this multi-objective mixed optimization problem, we propose the heuristic method to solve it. However, the traditional heuristic methods (e.g. genetic algorithm, greedy algorithm) are suitable for solving the problem within two optimization objectives. Therefore, we propose a new method based on the improved simulated annealing algorithm. Simulated annealing [16] is a popular choice for finding an optimum of the problems that have large search space, so we select simulated annealing. The main idea of simulated annealing is to accept solutions that are worse than the current one with some probability. A specific parameter named as temperature is proposed to control this process. The probability of accepting worse solutions decreases with temperature. The probability of accepting worse solutions decreases with the difference between objectives of current and new solution.

However, the original simulated annealing easily falls into the local minimum and loses the current optimal solution due to probabilistic choice. Therefore, a new algorithm, named State-aware

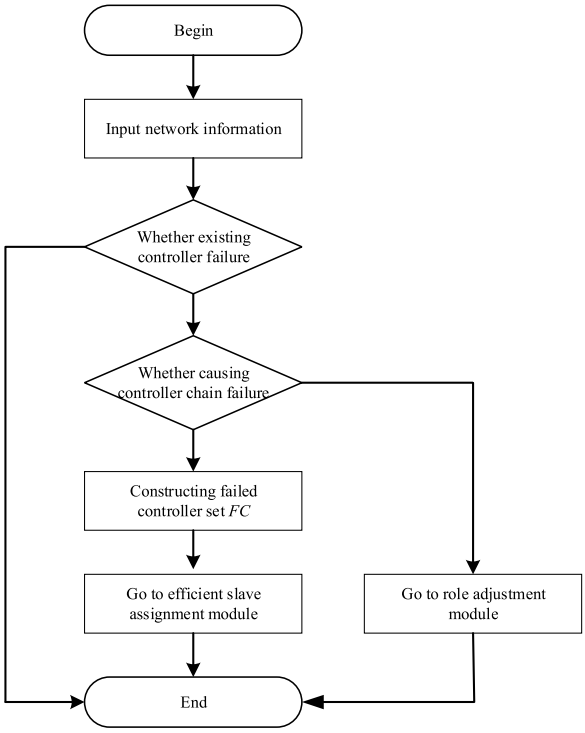


Fig. 6. The workflow of state detection module.

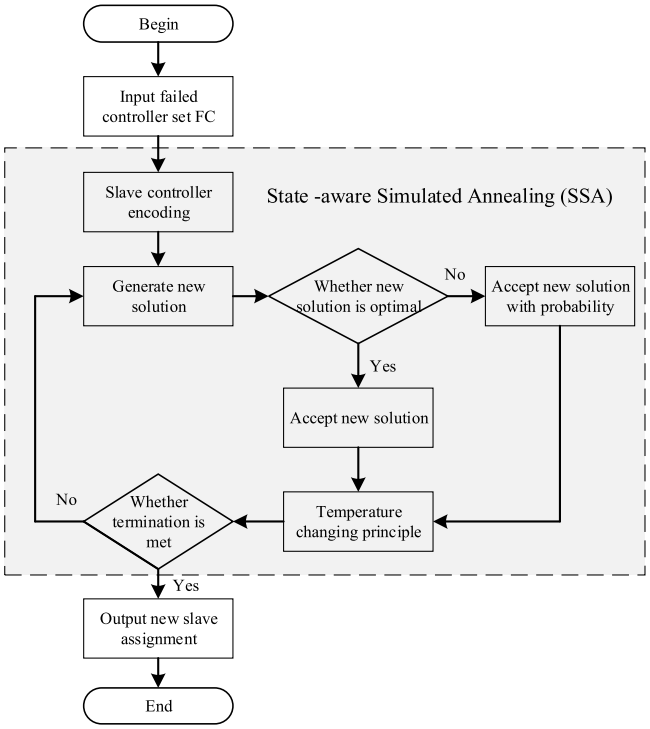


Fig. 7. The work flow of efficient slave assignment module.

FS	s_1	s_2	s_3	s_4	s_5
SA_1	0	1	0	0	1
SA_2	1	0	1	1	0

Fig. 8. An example of slave controller encoding.

Simulated Annealing (SSA), is proposed to efficiently solve the SCA problem. The improvement of SSA mainly includes two aspects. For one thing, we propose an efficient slave controller encoding based on the characteristics of slave controller assignment to improve operating efficiency of algorithm. For another thing, we design the new principle of temperature changing to adjust temperature to warming or cooling state, which can avoid the local optimum. In Fig. 7, we present the workflow of efficient slave assignment module, which includes SSA algorithm shown in gray rectangle field.

4.3.1. Slave controller encoding

The objective of efficient slave assignment module is to generate a new slave controller assignment for those switches whose master controller fails. Therefore, it is necessary to design an efficient encoding way for failed controllers and its managed switches to improve the efficiency of SSA algorithm. Failed controller set is $FC = \{c_1, c_2, \ldots, c_r\}$, where r is the number of failed controllers. The switches managed with failed controllers are set as $FS = \{s_1, s_2, \ldots,\}$. New slave assignment is $SA = \{SA_1, \ldots, SA_k, \ldots, SA_{M-r}\}$, and SA_k is the slave assignment scheme of the k^{th} normal controller, as shown in Eq. (25).

$$SA_k = \{s_1(k), s_2(k), \ldots, s_{|FS|}(k)\} \tag{25}$$

For convenience, we use a set of binary numbers to encode slave controller assignment, where $s_i(k) = 1$ represents the controller c_k is assigned to s_i as new slave. For example, Fig. 8 shows $FS = \{s_1, s_2, s_3, s_4, s_5\}$ and $SA = \{SA_1, SA_2\}$. The value of SA_1 in s_2 is equal to 1, which represents that controller c_1 is assigned to switch s_2 as the new slave.

4.3.2. Temperature changing principle

The temperature changing is an important feature of simulated annealing. The original simulated annealing algorithm only implements temperature cooling to accept or reject new solution. However, this way may plunge into local optima. Thus, we propose new temperature changing principle, including the process of temperature warming and cooling.

Temperature changing principle. During the iteration of SSA algorithm, if the value of fitness function (objective function) always keeps local maximum, the temperature rises again but always lowers than the initial temperature. Otherwise, if its value keeps local minimum, the temperature continues to fall. Moreover, we introduce the warming factor a and cooling factor b, as shown in Eq. (26) and Eq. (27), where T is temperature and I is the number of iterations.

$$T_I = a \cdot T_{I-1} \quad (1 < a) \tag{26}$$

$$T'_I = b \cdot T'_{I-1} \quad (0 < b < 1) \tag{27}$$

4.3.3. State-aware simulated annealing (SSA) algorithm

Based on the above preparations, we implement SSA algorithm. SSA takes the failed controller set FC and the annealing parameters as inputs and returns the new slave controller assignment to the next module. Further, the annealing parameters contain initial temperature T_0, number of iterations $I_{\max}$, the warming factor a, cooling factor b and final temperature T_f.

SSA starts with an initial temperature and reduces gradually until arriving at final temperature. SSA firstly encodes slave controller assignment and initializes the temperature and iteration (Line 1–2). Then, SSA randomly selects a solution as the starting

Table 2
State-aware simulated annealing algorithm.

Algorithm 1 State-aware simulated annealing (SSA) algorithm

Input: FC, T_0, T_f, $I_{\max}$, a, b
Output: NSA
1: Encode slave controller assignment
2. $T = T_0$, $i = 1$
3: Generate initial solution SA
4: Compute $\Gamma(SA) = \min(D^{slave}, |LV - LV_{failure}|, Z - R)$
5: ***while*** $T \geq T_f$ do
6: ***if*** Feasibility(SA) $= 1$ then
7: $NSA = SA$, $\Gamma(NSA) = \Gamma(SA)$
8: ***endif***
9: Get the new solution SA'
10: Compute $\Gamma(SA')$
11: $\Delta = \Gamma(SA') - \Gamma(SA)$
12: Generate p between 0 and 1
13: ***if*** $P(SA') \geq p$ then
14: $SA' = SA$, $\Gamma(SA') = \Gamma(SA)$
15: ***endif***
16: i++
17: ***if*** I_{max} iterations are performed at T
18: Implement temperature changing principle, and $i = 1$
19: ***endif***
20: ***endwhile***
21: Return new slave assignment NSA

Table 3
Efficient role adjustment algorithm.

Algorithm 2 Efficient Role Adjustment (ERA) algorithm

Input: OSA, NSA
Output: s_n^m and $s_n(k)$ for each switch, new network state
1: Accept outputs of state detection module and efficient slave assignment module
2. Determine type: OSA? NSA? $OSA \cup NSA$?
3: ***switch*** (type)
4: ***case*** OSA
5: ***while*** $(OSA \neq \phi)$
6: Changing controller's role $s_n^m = 1$ and $s_n(m) = 0$
7: ***endwhile***
8: ***case*** NSA
9: ***while*** $(NSA \neq \phi)$
10: Changing controller's role $s_n^m = 1$ and $s_n(m) = 0$
11: ***endwhile***
12: ***case*** $OSA \cup NSA$
13: ***while*** $(OSA \neq \phi \cup NSA \neq \phi)$
14: Changing controller's role $s_n^m = 1$ and $s_n(m) = 0$
15: ***endwhile***
16: Update network state to state detection module and data plane
17: Return new network state

point (Line 3) and computes its objective values $\Gamma(SA)$ based on Eqs. (15)–(16). Specifically, $\Gamma(SA)$ is composite value, considering D^{slave}, $|LV - LV_{failure}|$ and Z-R (Line 4). Here, we can adopt a solution obtained by greedy method as the start. Each solution is a set $SA = \{SA_1, \ldots, SA_k, \ldots, SA_{M-r}\}$, where $m - r$ is the number of controllers under the active states. Further, SSA searches the best feasible solution so far and its objectives. A solution is recorded if its objectives are less than the last solution. In each iteration, a random neighbor SA' of the current solution SA is generated and computed its objective values (Line 9–10).

According to Eq. (28), we accept the new solution with probability P. If the new solution is rejected, we change the temperature and generate another solution. The process will continue until the temperature arrives at the final temperature. The probability of accepting a solution is a function of temperature. There are two cases in terms of new solutions. On the one hand, if the new solution is better than current one, it will be accepted due to $e^{(-\Delta/T)} > 1$. On the other hand, if the new solution is worse than the current one, it will be accepted unless $e^{(-\Delta/T)} \geq p$, where Δ is the difference between objectives of new and current solutions, and p is a number between 0 and 1. We consider the two solutions are neighbors if they differ by at most one switch. In order to ensure the steady state, the algorithm should iterate enough times. After ending iteration, we modified the temperature based on temperature changing principle (Line 18). Finally, SSA outputs a new slave assignment NSA (Line 21). The pseudo-code of SSA algorithm is shown in Table 2.

$$P(SA') = e^{(-\Delta/T)} = e^{(-(\Gamma(SA)-\Gamma(SA'))/T)} > p \tag{28}$$

For algorithm 1, its time complexity mainly depends on the initial temperature T_0, final temperature T_f and iterations $I_{\max}$. For each temperature level, the algorithm will iterate $I_{\max}$ times. Moreover, the time complexity of while loop in Algorithm 1 is $O(MN)$. Therefore, the time complexity of algorithm 1 is O(MNI).

4.4. Role adjustment module

Role adjustment module has two functions in DSCA. On the one hand, it is responsible for adjusting the roles of controllers (master or slave) based on the inputs of state detection module and efficient slave assignment module. On the other hand, it not only feedbacks the updated network state into state detection module, but also informs the updated network state into the switches of data plane. Obviously, on the basis of the workflow of DSCA shown in Fig. 5, if both original and new slave assignment exist, the original slave assignment sent by state detection module directly reaches role adjustment module, while the new slave assignment reaches later. However, no matter which types of slave assignment (original or new) are processed, the network state always needs to be updated. In order to improve the processing efficiency, we propose a combinational processing method, which implements updating state after both original and new slave assignment completed, as shown in Fig. 9. Therefore, compared with individual processing, the combinational processing could reduce the times of updating state once the original slave assignment and new slave assignment exists side by side.

Correspondingly, we design an efficient role adjustment (ERA) algorithm in Table 3. The input of algorithm is original slave assignment OSA and new slave assignment NSA. Firstly, we determine the type of input (Line 2). If input only contains OSA or NSA, we directly adjust the controller role, and output binary variables s_n^m and $s_n(k)$ for each switch (Line 4–11). Otherwise, we process OSA or NSA according to the order of arrival, and update the network state until both OSA or NSA are completed (Line 12–13). Moreover, we get the s_n^m and $s_n(k)$. Finally, the algorithm outputs the updated network state to data plane and state detection module, respectively. The pseudo-code is shown in Table 3.

In Algorithm 2, it contains three while loops. For each while loop, its time complexity is $O(K)$, where K is the number of controllers needed to change roles. Therefore, we can conclude that the time complexity of Algorithm is $O(K)$.

5. Simulation and evaluation

In this section, we evaluate the effectiveness of DSCA through simulations. Actually, we will discuss the simulation setting and results.

5.1. Simulation setting

Simulation experiments are designed to verify the performance of the proposed method. There are several settings for experiments as follows.

(1) Experiment platform

We select OpenDaylight [17] as the experimental controller and use Mininet [18] as the test platform. OpenDaylight supports

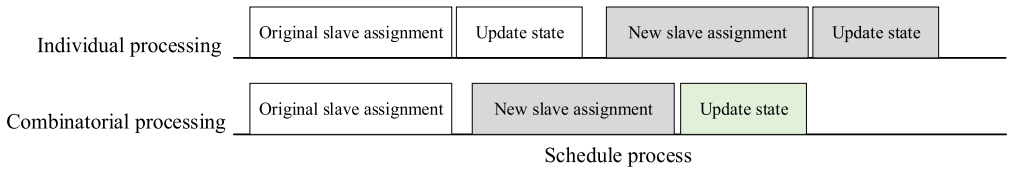


Fig. 9. Comparison of individual and combinational processing in role adjustment module.

Table 4
The characteristics of network topologies.

Topology	ABILENE	BELNET	CHINACOM
Node	11	21	42
Link	14	31	66

multiple versions of OpenFlow protocols. Both OpenDaylight and Mininet run on servers in the form of virtual machines. Specifically, the configuration of server includes Intel Core i5, 3.1 GHz and 8 GB RAM. The operation system is Ubuntu 16.04. Further, DSCA, the proposed solution, is loaded on OpenDaylight as an application.

(2) Topology setting

We select the authoritative network topologies from Topology Zoo [19] to make the experiments more persuasive. Topology Zoo is a project to collect data network topologies from around the world. It currently has over two hundred and fifty networks. We adopt three topologies (ABILENE, BELNET, CHINACOM) from the Topology Zoo. These three topologies are the most used network topologies in related literature, and their topology information has been known to us. Besides, their topology scales gradually expand. The characteristics of the selected network topologies are shown in Table 4. We use the latitude and longitude information of the selected network to calculate latency between network nodes.

(3) Parameter setting

We adopt Iperf [20] to generate TCP flows to represent the network traffic. The flow requests generated by switches are between 200 to 500 kilo packets/s. Moreover, the maximal capacity of controller is 9×10^6 packets/s based on OpenFlow 1.2 protocol. Due to the limitation of link bandwidth, we set the number of switches controlled by each controller is between 5 to 20 [21]. The algorithm file of the multi-objective mixed optimization is generated using MATLAB and sent it to CPLEX optimizer to solve [22].

(4) Compared schemes

Static: All switches in a domain select the same fixed controller as slaves. (e.g. switches controlled with controller 1 only select controller 2 that has minimum hop as slave)

Round: All switches in a domain select other controllers as slaves according to the predetermined slave controller order. (e.g. switches controlled with controller 1 select controller 2, 3, 4, 5 as slave in turn).

DSCA: All switches in a domain dynamically select different controllers as their slave controllers using the scheme present in Section 4.

5.2. Simulation results

5.2.1. Latency

Here, the latency mainly refers to the latency between switch and controller, as shown in Eq. (3). The worst case latencies of DSCA and compared schemes while considering controller failures are presented in Fig. 10. We have deployed 12 controllers in the network in advance, and the maximum number of failed controllers does not exceed the half of deployed controllers to make the experimental data be meaningful for comparison. This is because the latencies of three schemes will be extremely large if all controllers fail in the network. Thus, we vary the number of failed controllers from 0 to 6 for three networks, and the results are shown in Fig. 10(a) to Fig. 10(c). We can see that compared with the other two schemes, DSCA keeps worst case latency in a low level as the number of failed controllers increases.

The reasons can be explained as follows. When there is no controller failure in the network, all schemes have the same experimental results. With the increasing of the number of failed controllers, the worst case latency of Static sharply increases due to lack of planning for controller failures. Once a controller failure brings about controller chain failure, the controller performance cannot be effectively guaranteed and the worst case latency increases significantly. Though Round scheme designs a polling assignment mechanism for slave controller, the polling order just depends on the capacities of controllers. For Round scheme, if the slave controller that has the large hop is assigned to switch as new master, the worst case latency is still in a high level. Since the one of objectives of DSCA is to minimize the latency between switch and controller, it will plan ahead for controller failure and dynamically assign slave controllers to switches considering the latency constraint. Once detecting controller failure in the network, DSCA will transform the corresponding controllers from slave to master role. DSCA is less affected by controller failures, and its worst case latency averagely decreases 35.1% compared with Static and Round.

Because the worst case latency could not present the distribution of latencies between switches and controllers, we further compute the cumulative distribution function (CDF) of latencies under controller failure scene in Fig. 11. No matter in ABILENE, BELNET or CHINACOM, we can see that the maximum latency between switch and controller of DSCA is lower than the other two schemes'. Therefore, we conclude that the latency of DSCA is better than Static and Round in case of controller failure. In other words, DSCA has a better fault-tolerant performance.

5.2.2. Controller load balancing

In the multi-controller network, all controllers have different loads, while the more balanced the distribution of controller loads, the better the performance of control plane. Thus, we compute Root Mean Squared Error (RMSE) in Eq. (29) of all controllers based on Eq. (5) and Eq. (8) to evaluate the load balancing performance of three schemes. L_m and $L_m^{'}$ are the controller loads before and after controller failure, respectively.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{m}(L_m - L_m^{'})}{m}} \tag{29}$$

Here, a smaller RMSE means a better performance of load balancing. If all controllers have the same loads, the value of RMSE is 0. Fig. 12 describes controller RMSE value for three schemes, and r represents the number of failed controllers in network. Here we set $r = 1$, $r = 2$, and $r = 3$, respectively. In Fig. 12, the box indicates the center half of the data. It is clearly seen that DSCA has the lowest RMSE value, Round is the second, and Static is the highest. Meanwhile, we also observe that DSCA's RMSE is less influenced by the increment of number of failed controllers. Round's RMSE has the greater fluctuant range and it performs well if and only if $r = 1$. There are three reasons. Firstly, because

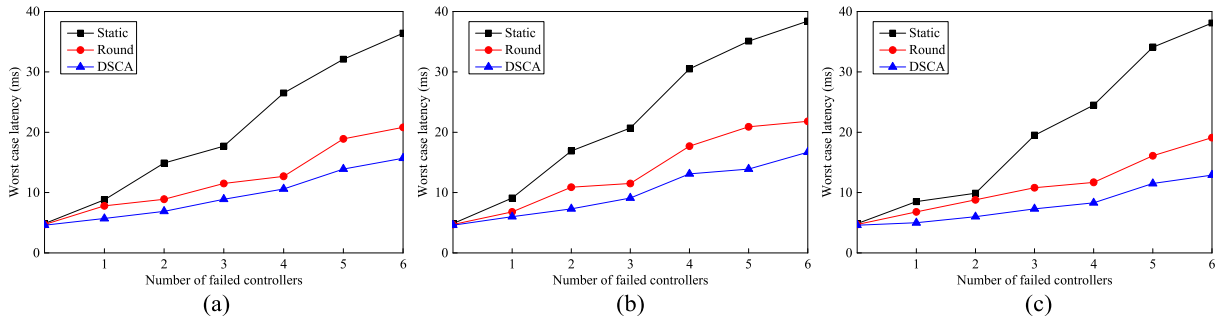


Fig. 10. Worst case latency of various schemes in three networks: (a) ABILENE; (b) BELNET; (c) CHINACOM.

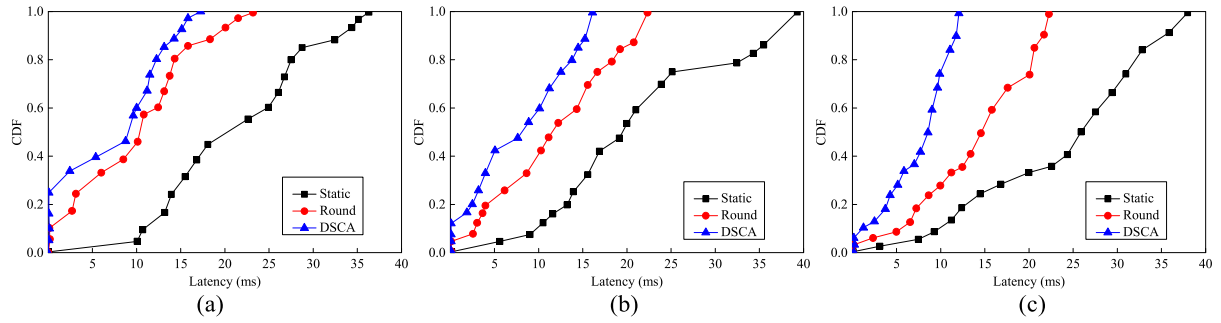


Fig. 11. CDF of latency of various schemes in three networks: (a) ABILENE; (b) BELNET; (c) CHINACOM.

Static scheme fixedly assigns slave controllers to switches without considering controller loads, Static's RMSE is always in a higher value under the controller failure scenario. Secondly, when there is only one failed controller in the network, Round can search the rational slave controllers for the switches of the failed controller by polling. If the number of failed controllers increases, Round will lose efficiency because the predetermined polling order is broken. Thirdly, benefiting from the efficient slave module and feedback process design, compared with the other two schemes, DSCA can dynamically adjust the assignment of slave controllers according to the failure condition, and ensure the load balancing performance of controller under the situation of controller failures. Hence, under different failure scenarios ($r = 1, 2, 3$), DSCA can always keep its controller RMSE in a lower level and achieve controller load balancing in case of controller failures.

5.2.3. Network crash probability

The objective of this paper is to improve the fault-tolerant performance of control plane. Intuitively, the more controller failures the network bears, the better its fault-tolerant performance. Therefore, in this experiment, we observe network crash probability p_{crash} caused by controller failures, which can be computed in Eq. (30). M^* and M represent the number of failed controllers and total controllers, respectively.

$$p_{crash} = \frac{M^*}{M} \tag{30}$$

Fig. 13 shows the results of three schemes in different networks. Analogizing experiment 1, we vary the number of failed controllers from 0 (minimum) to 12 (maximum), and observe the changing of network crash probability. It is clearly seen that DSCA outperforms Static and Round consistently, and also substantially, for different failure scenes but the extremes 0 and 1. More importantly, the curves representing DSCA have a slower growth rate than Static and Round. When there is no controller failure in the network, all schemes behave similarly. As the number of failed controllers increases to 5, the probability of network crash for DSCA is less than half of that for Round. Specifically, DSCA has stayed below 0.2 in different networks averagely, while Round has increased above 0.8. Afterwards, though the differences between all schemes diminish, Fig. 13 shows that DSCA reaches near 100% of network crash probability later than both Static and Round schemes.

The above results can be explained as follows. Static does not consider the instance of insufficient capacity of the nearest controller, which is likely to produce the controller chain failure. Thus, the network crash probability of Static grows quickly with the increment of failed controllers. Round performs better than Static scheme when there are a few failed controllers in the network. However, the stiff polling order of Round will lose efficiency if existing plenty of failed controllers. Compared with Static and Round, DSCA plans head for controller failure and dynamically executes efficient slave controller assignment. Therefore, DSCA can suffer more failed controllers and achieve high robust and fault-tolerant control plane.

5.2.4. Performance of heuristic

The core of DSCA is the efficient slave assignment module, which is developed by heuristic method. In order to show the performance of improved simulated annealing, we set original simulated annealing as comparison to evaluate latency optimization and running time. The settings of basic parameters are as follows. The range of temperature is from 0.01 to 20. We set the iteration $I_{max} = 300$, and warming factor $a = 1.05$ and cooling factor $b = 0.95$. Specifically, the original simulated annealing only has a cooling factor. We execute each instance for 50 times and compute the average result for data reliability. Fig. 14 shows the latency optimization and running time results of two methods in CHINACOM network. We can see that the improved simulated annealing proposed by this paper outperforms better than the original simulated annealing in terms of latency optimization, while the difference of running time of these two methods is small. The reasons are as follows. We improve simulated annealing algorithm by designing the specific slave controller encoding and temperature changing principle to speed up the execution efficiency and avoid the local optimal. Though the running time of the improved

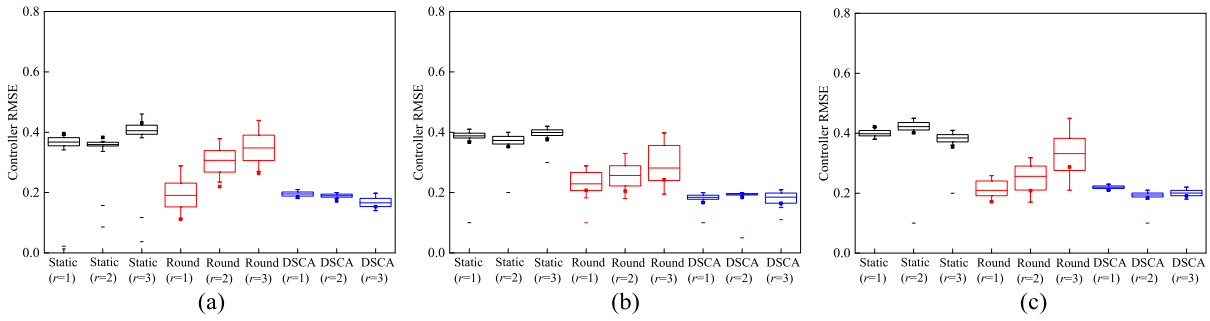


Fig. 12. Controller RMSE of various schemes in three networks: (a) ABILENE; (b) BELNET; (c) CHINACOM.

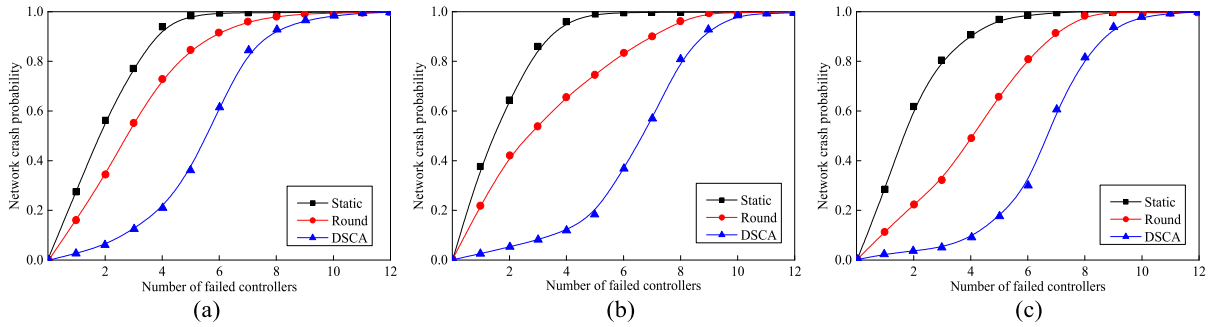


Fig. 13. Network crash probability of various schemes in three networks: (a) ABILENE; (b) BELNET; (c) CHINACOM.

method is slightly higher than the original method's, it brings a higher optimization benefit such as greatly reducing latency under controller failures. Moreover, the performance of algorithm can be further improved by increasing iterations and adjusting temperature parameters.

6. Related work

There are several researches on the fault-tolerant control plane, which could be divided into two aspects: reliable control plane and failover design of control plane.

6.1. Reliable control plane

In the initial SDN design, there is only one single controller in the network. For coping with the large-scale network, some researchers propose to deploy multi-controller, which can be divided into two aspects: flat architecture (e.g., HyperFlow [3], Onix [4]), and hierarchical architecture (e.g., Kandoo [5]). The purpose of multi-controller not only improves the scalability of SDN, but also increases the reliability of control plane. Based on this proposal, several works investigate the reliable control plane combined with multi-controller. Hu et al. [23] define a new metric, named expected percentage of control path loss, to characterize the reliability of SDN. Meanwhile, Reliability-aware Controller is demonstrated as an NP-hard problem, and several placement algorithms are examined to solve this problem. Lucas et al. [24] propose Survivor, a controller placement strategy, to address the single path between switch and controller failure. Survivor correctly explores the path diversity problem and considers capacity-awareness proactively for controller deployment to avoid controller overload. However, it is lack of considering multi-controller failures scene. Song et al. [25] study the reliability of SDN from the perspective of control path. They mainly focus on the reliability challenges in a control path network between control layer and data layer, and design control path reliability algorithms and a novel control message classification and prioritization system to enhance the reliability of SDN. Jiménez et al. [26] design an algorithm called K-Critical that places controllers to achieve a robust control. K-Critical discovers the minimum number of controllers and their locations to create a robust control topology that deals robustly with failures and balances the load among the selected controllers. Stanislav et al. [27] introduce POCO, a framework for Pareto-based Optimal COntroller placement, that provides operators with Pareto optimal placements with respect to controller failure, isolated node, load balancing and inter-controller latency. The availability of POCO is analyzed in detail via an evaluation featuring numerous real-world topologies. Killi et al. [28] formulate a mathematical model for the capacitated controller placement that aims to reduce the worst-case latency between switches and controllers to deploy the limited number of controller. Meanwhile, the authors also introduce a variant of the proposed model that minimizes the worst-case latencies with and without failure together. The above works mainly focus on researching how to deploy multi-controller to improve the reliability and avoid failure, but they are static and inflexible.

6.2. Failover design of control plane

It is clearly seen that stiff and fixed control plane design cannot gurantee the real-time reliability of controllers under the network with drastic traffic. Thus, several works propose to build failover mechanism for control plane to achieve fault tolerance. Neda et al. [29] design fast failover protection for control traffic once there are disconnections between switches and the controllers. By maximizing the possibility of fast failover, the authors can achieve resilience-aware control-traffic routing. After the OpenFlow 1.2 protocol has been proposed, the researchers propose to use both master and slave controllers to implement failover. Dixit et al. [30] propose ElastiCon, as a new controller architecture, to periodically monitor the load on each controller, detect imbalance and failure, and automatically balance the load across controllers by migrating switches. Meanwhile, in order to harmonize the migration, a novel switch migration protocol is

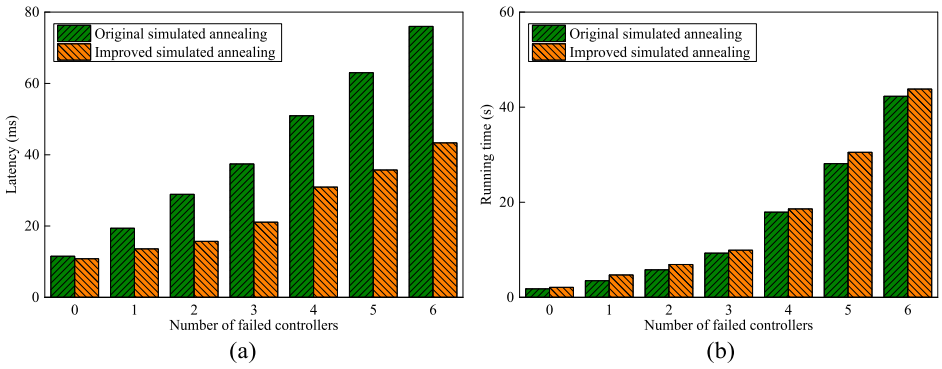


Fig. 14. Performance of improved simulated annealing on CHINACOM: (a) Latency (b) Running time.

designed for enabling such as load shifting, which conforms to the OpenFlow standard. Gonzalez et al. [31] design a new mechanism considering consistency and fault tolerance of SDN controllers. Its objective is to bring the performance of SDN Master–Slave controller as close as possible to the one offered by single controller, regardless of whether there is controller failure. Through introducing a simple replication scheme, this mechanism executes consistency and correction check, which influences the network performance only during a few slots. Liang et al. [32] propose to construct a scalable and crash-tolerant load balancing architecture, which can dynamically shift the load across the multiple controllers by switch migration. Moreover, this architecture also supports controller failover without switch disconnection avoiding the single point of failure problem. The prototype system is implemented in the OpenDaylight controller. Xie et al. [33] research minimal fault-tolerant coverage of controllers in the data center environment. They solve this problem from three aspects: minimal coverage, minimal fault-tolerant coverage, and the minimal communication overhead among controllers. Correspondingly, the efficient algorithms are designed to achieve those objectives. However, it does not consider the dynamic traffic in the network. Based on the above work, we can observe that the existing researchers mostly design backup and failover mechanism to protect control plane, but they focus on single controller failure and are lack of overall planning for failures.

7. Conclusion

In this paper, we attempt to improve the fault-tolerant performance of control plane from the perspective of slave controller assignment and propose a dynamic slave controller assignment (DSCA) scheme. Firstly, we describe controller chain failure phenomenon caused by controller failure, which may crash the entire network. Through analyses, we find that rational slave controller assignment is the key to guarantee the fault-tolerant performance of control plane. Then, we formulate the SCA problem, considering latency, load balancing and robustness, as a multi-objective mixed optimization. Further, DSCA is proposed to effectively solve the SCA problem, and it can be divided into three modules including state detection, efficient slave assignment and role adjustment. More importantly, the improved heuristic algorithms are loaded on the corresponding modules to increase efficiency. Simulation results show that, compared with other schemes, DSCA can effectively ensure the fault-tolerant performance of control plane and minimize the impact of controller failure on the entire network.

In future work, we plan to provide more insights into the theoretical model and extend DSCA scheme to a large-scale network with more real traffic to evaluate its performance.

Acknowledgments

This work is supported by the Project of National Network Cyberspace Security, China (Grant No. 2017YFB0803204), Foundation for Innovative Research Group of National Natural Science Foundation of China (Grant No. 61521003), National Natural Science Foundation of China (Grant No. 61502530, 61802429, 61872382).

References

[1] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, Jonathan Turner, OpenFlow: Enabling Innovation in Campus Networks, ACM SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74.
[2] T. Wang, F. Liu, J. Guo, H. Xu, Dynamic SDN controller assignment in data center networks: Stable matching with transfers, in: Proc. IEEE INFOCOM 2016, IEEE International Conference on Computer Communications IEEE, 2016, pp. 1–9.
[3] T. Hu, P. Yi, Z. Guo, J. Lan, J. Zhang, Bidirectional Matching Strategy for Multi-Controller Deployment in Distributed Software Defined Networking, IEEE Access 6 (2018) 14946–14953.
[4] I.F. Akyildiz, A. Lee, P. Wang, M. Luo, C. Wu, A roadmap for traffic engineering in SDN-OpenFlow networks, Comput. Netw. Int. J. Comput. Telecommun. Netw. 71 (3) (2014) 1–30.
[5] D. Dotan, R.Y. Pinter, HyperFlow: An integrated visual query and dataflow language for end-user information analysis, in: Proc. 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05), 2005, pp. 27–34.
[6] S.H. Yeganeh, Y. Ganjali, Kandoo: A framework for efficient and scalable offloading of control applications, in: Proc. 1st Workshop HotSDN, 2012, pp. 19–24.
[7] Z. Guo, M. Su, Y. Xu, Z. Duan, L. Wang, S. Hui, Improving the performance of load balancing in software-defined networks through load variance-based synchronization, Comput. Netw. 68 (2014) 95–109.
[8] M.F. Bari, A.R. Roy, S.R. Chowdhury, Q. Zhang, M.F. Zhani, R. Ahmed, Dynamic controller provisioning in software defined networks, in: Proc. International Conference on Network and Service Management IEEE, 2013, pp. 18–25.
[9] T. Hu, Z. Guo, P. Yi, T. Baker, J. Lan, Multi-controller Based Software-Defined Networking: A Survey, IEEE Access 6 (2018) 15980–15996.
[10] Brandon Heller, Rob Sherwood, Nick McKeown, The controller placement problem, ACM SIGCOMM Comput. Commun. Rev. 42 (4) (2012) 473–478.
[11] M.T.I.U. Huque, W. Si, G. Jourjon, V. Gramoli, Large-Scale Dynamic Controller Placement, IEEE Trans. Netw. Serv. Manag. 14 (1) (2017) 63–76.
[12] M. He, A. Basta, A. Blenk, W. Kellerer, Modeling flow setup time for controller placement in SDN: Evaluation for dynamic flows, in: Proc. IEEE International Conference on Communications IEEE, 2017.
[13] K. Kuroki, M. Fukushima, M. Hayashi, N. Matsumoto, Redundancy method for highly available openflow controller, Int. J. Adv. Internet Technol. (2014) 114–123.
[14] N. Beheshti, Y. Zhang, Fast failover for control traffic in software-defined networks, in: Proc Global Communications Conference, 2013, 2665-2670.
[15] T. Wang, Z. Guo, H. Chen, W. Liu, BWManager: Mitigating Denial of Service Attacks in Software-defined Networks through Bandwidth Prediction, in: IEEE Transactions on Network and Service Management, 2018, http://dx.doi.org/10.1109/TNSM.2018.2873639.
[16] David S. Johnson, Cecilia R. Aragon, Lyle A. McGeoch, Catherine Schevon, Optimization by Simulated Annealing: An Experimental Evaluation, At & t Bell Labs, 1991, pp. 215–226.

[17] J. Medved, R. Varga, A. Tkacik, K. Gray, OpenDaylight: Towards a model-driven SDN controller architecture, In Proc. IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks. 2014, pp. 1–6.
[18] Kaur. Karamjeet, J. Singh, N.S. Ghumman, Mininet as software defined networking testing platform, In Proc. International Conference on Communication Computing & Systems, 2014.
[19] S. Knight, H.X. Nguyen, N. Falkner, R. Bowden, M. Roughan, The Internet Topology Zoo, IEEE J. Sel. Areas Commun. 29 (9) (2011).
[20] Ioan Raicu, Catalin Dumitrescu, Matei Ripeanu, Ian Foster, The Design Performance Use of DiPerF: An automated DIstributed Performance evaluation Framework, J. Grid Comput. 4 (3) (2006) 287–309.
[21] G. Cheng, H. Chen, Z. Wang, S. Chen, DHA: Distributed decisions on the switch migration toward a scalable SDN control plane, in: Proc. Ifip NETWORKING Conference IFIP, 2015, 471-477.
[22] IBM ILOG CPLEX, 2015, Accessed on Jul. 2015 [Online] Available: http://www.01ibm.Com/software/integration/optimization/cplex-optimizer.
[23] Y. Hu, W. Wang, X. Gong, X. Que, On reliability-optimized controller placement for Software-Defined Networks, China Commun. 11 (2) (2014) 38–54.
[24] L.F. Muller, R.R. Oliveira, M.C. Luizelli, L.P. Gaspary, Survivor: An enhanced controller placement strategy for improving SDN survivability, in: Proc. Global Communications Conference IEEE 2014, pp. 1909–1915.
[25] S. Song, H. Park, B.Y. Choi, T. Choi, H. Zhu, Control Path Management Framework for Enhancing Software-Defined Network (SDN) Reliability, IEEE Trans. Netw. Serv. Manag. (2017) 1–18.
[26] Yury Jimenez, C. Cervello-Pastor, A.J. Garcia, On the controller placement for designing a distributed SDN control layer, In Proc. NETWORKING Conference IEEE, 2014, pp. 1–9.
[27] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, Heuristic Approaches to the Controller Placement Problem in Large Scale SDN Networks, IEEE Trans. Netw. Serv. Manag. 12 (1) (2015) 4–17.
[28] Bala Prakasa Rao Killi, S.V. Rao, Optimal Model for Failure Foresight Capacitated Controller Placement in Software-Defined Networks, IEEE Commun. Lett. 20 (6) (2016) 1108–1111.
[29] Neda Beheshti, Y. Zhang, Fast failover for control traffic in Software-defined Networks, in: Proc. Global Communications Conference IEEE 2013, pp. 2665–2670.
[30] Advait Abhay Dixit, S. Fang, T.V. Lakshman, R. Kompella, ElastiCon; An elastic distributed SDN controller, in: Proc. ACM SIGCOMM Workshop on Hot Topics in Software Defined NETWORKING ACM, 2013, pp. 7–12.
[31] A.J. Gonzalez, G. Nencioni, B.E. Helvik, A. Kamisinski, A Fault-Tolerant and Consistent SDN Controller, in: Proc. Global Communications Conference IEEE, 2017, pp. 1–6.
[32] Liang. Chu, R. Kawashima, H. Matsuo, Scalable and crash-tolerant load balancing based on switch migration for multiple open flow controllers, In Proc. Second International Symposium on Computing and NETWORKING IEEE, 2015, 171-177.
[33] J. Xie, D. Guo, X. Zhu, B. Ren, H. Chen, Minimal Fault-tolerant Coverage of Controllers in IaaS Datacenters, IEEE Trans. Serv. Comput. (2017) 1–17.

Tao Hu is currently a Doctor candidate at National Digital Switching System Engineering and Technological Research Center at Zhengzhou, China. He earned B.E. degree in the Xi'an Jiaotong University. He is currently pursuing the doctor degree in network cyber security. His research interests include software defined networking, control plane and network security. E-mail: hutaondsc@163.com

Peng Yi, is a research fellow of National Digital Switching System Engineering and Technological Research Center. His contributions encompass aspects of security, network architecture (SDN, CCN) and signal processing (Corresponding author, E-mail: yipengndsc@163.com).

Zehua Guo is a Research Associate at the Department of Computer Science and Engineering of University of Minnesota Twin Cities. He received B.S. degree from Northwestern Polytechnical University, M.S. degree from Xidian University, and Ph.D. from Northwestern Polytechnical University. He was a Research Fellow at the Department of Electrical and Computer Engineering, New York University Tandon School of Engineering. His research interests include software-defined networking, network function virtualization, data center network, cloud computing, content delivery network, network security, green network, machine learning, and Internet exchange.

Julong Lan is a professor of National Digital Switching System Engineering and Technological Research Center. His contributions encompass aspects of information theory and security, network architecture and signal processing.

Yuxiang Hu is an associate researcher of National Digital Switching System Engineering and Technological Research Center. His contributions encompass aspects of network architecture and routing.