Computer Networks 185 (2021) 107740

Contents lists available at ScienceDirect

Computer Networks

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

SQHCP: Secure-aware and QoS-guaranteed heterogeneous controller placement for software-defined networking

Peng Yi [a], Tao Hu [a,b,*], Yuxiang Hu [a], Julong Lan [a], Zhen Zhang [a], Ziyong Li [b]

[a] National Digital Switching System Engineering and Technological Research Center, China
[b] Information Engineering University, China

ARTICLE INFO

Keywords:
SDN
Heterogeneous controller placement
Security
QoS
Heuristic algorithm

ABSTRACT

As the large-scale application of software-defined networking (SDN), one major research challenge for the SDN is to place multi-controllers reasonably. The researchers have proposed a lot of good solutions to the controller placement problem (CPP). However, they usually chose the same type of controllers (called homogeneous controllers) and ignored the common-mode fault caused. To fill this gap, considering the heterogeneity of controllers and QoS, this paper first focuses on different types of controllers (heterogeneous controllers) and proposes a **He**terogeneous **C**ontroller **P**lacement **P**roblem (**HeCPP**) to minimize network delay, balance heterogeneous controller utilization, and lower control plane fault rate. We introduce a **S**ecure-aware and **Q**oS-guaranteed **H**eterogeneous **C**ontroller **P**lacement (**SQHCP**) approach to solve HeCPP effectively. Specifically, SQHCP consists of two steps. In step 1, the types and the number of heterogeneous controllers are determined based on dynamic planning to enhance the security of the control plane. After that, with the inspiration of the "divide and conquer" strategy, in step 2, the network is partitioned into several subnets based on the K-means clustering, and the genetic algorithm is improved to place the heterogeneous controller for each subnet to guarantee QoS. The theoretical analysis has proven the feasibility of SQHCP. Simulation shows that SQHCP not only degrades the control plane fault rate but also outperforms the existing approaches in terms of delays and load balancing.

1. Introduction

Software-Defined Networking (SDN), as a novel network paradigm, abstracts the network control function from the distributed data plane, and all control functions are coupled into a logically centralized entity, which is called the controller [1]. With the help of the above characteristics, SDN transforms the Internet into a programmable and manageable infrastructure. Nowadays, SDN has been widely used in many real-world networks such as Google B4 [2]. With the scale expansion of the SDN, it is difficult for the single controller to meet the requirements of delay and flow requests in the network [3]. In other words, the single controller cannot manage the large-scale network. Moreover, the physically centralized controller may encounter a single point of failure [3], degrading the reliability and availability of the SDN significantly. Once the single controller suffers from the performance bottleneck, the advantages of SDN will be lost.

In recent years, researchers have proposed several distributed multi-controller frameworks: Kandoo [4], Onix [5], DISCO [6], just to name a few. The multi-controllers manage the network in a logically centralized but physically distributed way, which improves the scalability and fault tolerance of the SDN network effectively [7]. As the number of controllers increases, a new set of problems arises, the most serious of which is how to place multiple controllers. Correspondingly, researchers have introduced various solutions to the controller placement problem (CPP) [8]. In fact, in terms of solving the CPP, simply optimizing the number of SDN controllers is inefficient, while controller locations and mapping relationships also affect the network performance dramatically. As first defined in [9], the objectives of CPP are to determine the locations of controllers so as to reduce the average and worst-case propagation latencies. Further, Yao et al. [10] introduced the capacity-aware CPP in the SDN. Several researchers considered multiple criteria, such as delay, link utilization rate, and controller loads, to address the CPP more comprehensively [11,12].

Nowadays, considerable research efforts have been made to address the CPP, and the existing studies generally place the homogeneous controllers in the SDN, which means that the entire network is managed

* Corresponding author.
E-mail addresses: yipengndsc@163.com (P. Yi), hutaondsc@163.com (T. Hu).

https://doi.org/10.1016/j.comnet.2020.107740
Received 15 July 2020; Received in revised form 16 November 2020; Accepted 12 December 2020
Available online 15 December 2020
1389-1286/© 2020 Elsevier B.V. All rights reserved.

by the same type of controllers. However, there are great potential security threats in the SDN control plane composed of the homogeneous controllers. Since the homogeneous controllers have the same designs, any possible vulnerabilities would be reflected in the identical components of homogeneous controllers [13]. If the attackers have mastered one vulnerability in the controller, they can exploit this vulnerability to attack all homogeneous controllers, which damages the entire control plane. Formally, the homogeneous controller common-mode fault is described as follows: in the multi-controllers SDN, if one controller fails because of one or more design vulnerabilities, all homogeneous controllers that have the same type with this controller will be threatened in the same way. More details about it will be shown in Section 2.

Although the researchers have proposed a great number of solutions to CPP [8] in the SDN, they may underestimate or even ignore the homogeneous controller common-mode fault. If the malicious attackers exploit the same vulnerability to activate the common-mode fault, the SDN control plane composed of homogeneous controllers is destroyed regardless of optimizing controller placement. Furthermore, as a special network operation system [14], the SDN controller is a software product essentially, and there are inevitable vulnerabilities in the process of its design [15].

With the basic axiom that there are usually multiple solutions to the same problem and multiple implementation structures for the same function [16], the heterogeneous controllers (e.g., NOX, Ryu, Floodlight) designed by different vendors can effectively address the homogeneous controller common-mode fault. For example, several researchers have developed heterogeneous controllers in the SDN [17–19]. Dixit et al. [17] have composed the heterogeneous SDN controllers with FlowBricks to provide all desired network services. [18] and [19] developed the heterogeneous controllers to cope with hijacking and modification attacks. In light of this observation, we consider placing the heterogeneous controllers in the network. We formulate a **He**terogeneous **C**ontroller **P**lacement **P**roblem (**HeCPP**) formally and propose a **S**ecure-aware and **Q**oS-guaranteed **H**eterogeneous **C**ontroller **P**lacement (**SQHCP**) approach to solve the HeCPP effectively. To the best of our knowledge, this paper is the first work for studying the heterogeneous controller placement in the SDN network. The main contributions of this paper are listed as follows:

- We analyze the problem of the homogeneous controller common-mode fault in the existing CPP solutions and explain the validity of the heterogeneous controllers for enhancing the security of the control plane by testing four well-known controllers.
- We define the HeCPP under the heterogeneous controller condition. The objective function of the HeCPP with multiple constraints is also formulated to minimize network delay, balance heterogeneous controller utilization, and lower control plane fault rate.
- We propose the SQHCP approach for solving the HeCPP, which consists of two steps. Step 1 identifies the types and the number of heterogeneous controllers based on dynamic planning. Step 2 partitions the network into multi-subnets based on K-means clustering and then improves the genetic algorithm to optimize the heterogeneous controller placement for each subnet.
- We analyze the feasibility of SQHCP with several theorems theoretically. Simulation shows that our proposal outperforms several representative approaches under different network topologies, which greatly enhances the security of the control plane and guarantees QoS.

The rest of the paper is structured as follows. Section 2 gives the motivation of this paper. HeCPP is formulated in Section 3. The proposed approach SQHCP is described in Section 4. Section 5 evaluates the performance of our proposal. Section 6 shows the related work. Section 7 concludes this paper.

2. Motivation

In this section, we first introduce the homogeneous controller common-mode fault in the existing CPP solutions. Then, we illustrate the effectiveness of heterogeneous controllers for enhancing the security of the control plane through testing the four universal controllers.

2.1. Homogeneous controller common-mode fault

Multi-controller ameliorates the resilience and fault tolerance of the SDN network. The existing CPP solutions generally consider that the control plane is composed of the homogeneous controllers, which means that the controllers placed in the SDN are the same type. The homogeneous multi-controller placement is the most extensive framework of the SDN at present and has several advantages in the practical application process: 1) The same type of controllers can communicate with each other without boundaries; 2) The homogeneous controllers are easy to be managed and configured; 3) the SDN network with homogeneous controllers has lower maintenance and investment costs. However, the homogeneous controller placement way has no advantage in SDN network security. The reason can be explained as follows. If attackers have mastered the vulnerability of one controller, they can exploit this vulnerability to launch malicious attacks, such as the message leak, disconnection with switches, or controller crash [15]. To make matter worse, due to the homogeneity of controllers placed, the other controllers will suffer from the same attack in the SDN. We call this phenomenon the homogeneous controller common-mode fault.

For example, in an SDN network, all controllers are homogeneous (e.g., Floodlight), each of which manages the corresponding SDN subnet. When the attackers have probed that the controller is Floodlight as well as mastered one vulnerability in the Floodlight (e.g., CVE-2014–2304 is an exposed vulnerability in OpenFlow protocol for Floodlight 0.9 version), they can exploit this vulnerability to crash it. No matter how many Floodlight controllers are placed as the backups in the control plane, the attackers can still easily destroy them through exploiting this vulnerability. At last, all Floodlight controllers fail in the same way, and the entire control plane is damaged by homogeneous controller common-mode fault.

As the core component of the SDN, the controller is the "brain" of the whole network. Nowadays, most controllers are static and have poor abilities to defend against probing attacks [20]. As discussed earlier, the existing CPP solutions usually place the homogeneous controllers in the SDN, all of which have the same vulnerabilities and are incapable of coping with the common-mode fault threat. What calls for special attention is that all controllers are software products that inevitably have vulnerabilities in their designs [14,15]. Actually, the common-mode fault is not unique to SDN controllers but widely exists in industrial productions and control systems [21], while the most general solution is the principle of heterogeneity [22]. The probability of the common-mode fault can be reduced effectively by using multiple heterogeneous components with equivalent functions.

Therefore, the heterogeneous controllers have the advantage of ensuring the SDN security in terms of homogeneous controller common-mode fault. However, it should be noted that there are following limitations when heterogeneous controllers are adopted: 1) Heterogeneous controllers are developed in different programming languages and have high requirements for the east-west interface of the controller; 2) Heterogeneous controllers require the coordination mechanism to synchronize network state (e.g., flow table, topology view) to ensure consistency.

Now, there are already proven solutions to the above limitations of heterogeneous controllers. For example, the authors in [19] improve ZeroMQ, a message communication tools for distributed systems, to implement the functions of the east-west interface for heterogeneous controllers. Moreover, distributed control plane like HyperFlow [51] is proposed to synchronize network-wide views of controllers based on the publish/subscribe messaging paradigm. Thus, although the heterogeneous controller placement has some limitations in practice, it is technically feasible by reference to the existing work and solutions.

2.2. *Security of control plane composed of heterogeneous controllers*

Inspired by the above, we consider that the key to solving the homogeneous controller common-mode fault is heterogeneous controller placement. Fortunately, as SDN technology evolves, there are more than 30 types of controllers, which are developed by different vendors/universities/institutes and programmed in different languages [23]. Therefore, the controllers with diversified technical features are sufficient to meet the requirement of the heterogeneity in the controller placement [18,19,24].

In the SDN, each type of controller (e.g., NOX, Ryu, Floodlight, ONOS) can complete network control and management tasks independently, indicating that they are equivalent in functions. In practice, there are different vulnerabilities in all kinds of heterogeneous controllers, but the heterogeneous environments and designs (e.g., NOX, Ryu, and Floodlight controllers are programmed in $C++$, Python, and Java, respectively) make the trigger mechanism for most controller vulnerabilities different. In some special cases, one vulnerability may exist in different types of controllers. However, a more general conclusion is that the same vulnerability rarely occurs in heterogeneous controllers written by different programming languages [19]. Theoretically, the heterogeneous controllers are independent of each other and have no dependencies during runtime. It is difficult for the attackers to make all heterogeneous controllers show abnormal behaviors at the same time through exploiting the same vulnerability [19].

In light of the above understanding, we propose, for the first time, to consider the heterogeneity of controllers into the CPP to address the homogeneous controller common-mode fault in the existing solutions. To this end, this subsection tests different types of controllers and explains that heterogeneous controllers can indeed weaken the controller common-mode fault. Further, the test results also provide a reference for subsequent heterogeneous controller placement. Now, several researchers have evaluated various open-source controllers, but they only compare the performances of controllers [24,25] (e.g., Packet_In response time, flow setup time). Distinguished from existing studies, we compare the security characteristics of four SDN controllers, and the test details are described as follows.

Test objects. The tested SDN controllers consist of NOX (programmed in $C++$, single thread), Ryu (programmed in Python, single thread), Floodlight (programmed in Java, multiple threads), and ONOS (programmed in Java, multiple threads).

Test environments. The testbed consists of two servers (Intel E3–1230, 4 Cores, 2.3 GHz, 32 GB RAM, Ubuntu 16.04) connected with one physical link (10 Gbps). One server uploads the controller and Mininet, while the other server generates traffic based on specific test cases. Two independent servers are used to guarantee that traffic generation does not affect controller performance. We use hcprobe tool to test the SDN controllers and record the results.

Test cases. In order to evaluate the security of heterogeneous controllers in general, we check their performances under various attacks, which are represented by incorrect OpenFlow messages. This is because the adversaries usually attack the controller through the OpenFlow channel [7]. We assess the controllers through the following cases. **Attack case 1**, incorrect OpenFlow message length. Packet_In messages with non-normative length fields are created in the OpenFlow header. **Attack case 2**, invalid OpenFlow protocol version. All controllers only support OpenFlow 1.2 version, which is declared when the switch and controller establish OpenFlow communication, but Packet_In message is set up an invalid OpenFlow protocol version. **Attack case 3**, incorrect OpenFlow message type. The values of the header fields do not correspond to the types of encapsulated OpenFlow messages. **Attack case 4**, incorrect protocol type. The code of the ARP protocol (0×0806) is put in the Ethernet header.

Test results. The results can be categorized into four grades (**A, B, C, D**) according to its performance: **A**) The controller defenses the attack effectively and runs normally; **B**) The controller fails to identify the attack and there is a potential vulnerability, but it still works; **C**) The controller disconnects with switches; **D**) The controller crashes. We run experiments 50 times to eliminate the random error. The results are shown in Fig. 1.

From Fig. 1, we can draw a conclusion that the same attack only threatens certain types of controllers, while the other types of controllers are unaffected or less vulnerable. For example, when attacked by the incorrect protocol type, ONOS crashes, while NOX, Ryu, and Floodlight are not affected. Note that some specific bugs may exist in multiple different controllers (e.g., the incorrect OpenFlow message length makes both NOX and Floodlight disabled), but the control plane can still keep active if the types of heterogeneous controller deployed are sufficiently rich (e.g., not only NOX, Floodlight but also Ryu, ONOS). Therefore, the reasonable combination of heterogeneous controllers can mitigate homogeneous controller common-mode fault and improve the security of the control plane.

3. Problem formulation

This section first defines several notations for modeling the SDN network and then introduces three metrics (end-to-end delay, controller utilization rate, controller fault rate) considered for heterogeneous controller placement. Finally, we formally formulate the HeCPP in this section.

Table 1 and Table 2 provide an overview of the notations and variables used in this paper, respectively.

3.1. *Network modeling*

SDN network is represented as an undirected graph $G = (V,L)$, where V is a set of N nodes and L is a set of bidirectionally physical links between nodes. Each node v_i corresponds to an SDN switch s_i (Note that v_i and s_i are used interchangeably in this paper). S is a set of switches and $|S| = |V| = N$. $h(v_i, v_j)$ indicates the minimal packet propagation delay between two nodes v_i and v_j. Binary variable l_{ij} is the connecting relation between nodes v_i and v_j, where $l_{ij} = 1$ represents that there is a direct connection between nodes v_i and v_j, as shown in Eq. (1). Moreover, we assume that the network traffic tends to be stable in the SDN network [26], and there is little difference in the traffic change in each time period. Besides, flow requests generated by every local switch s_i follow Poisson distribution at the rate of λ_i [27].

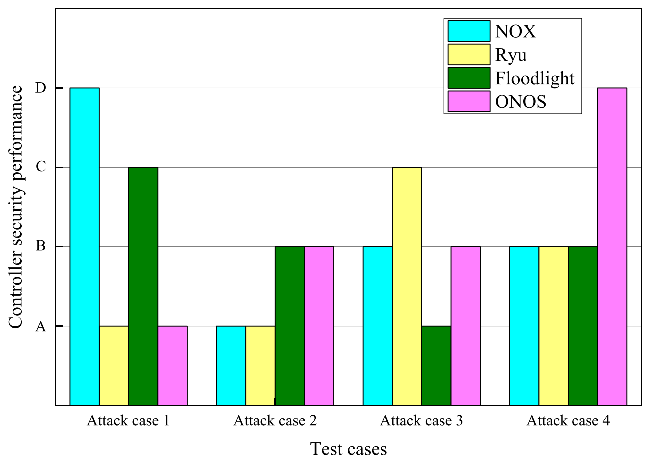

Fig. 1. The security performance of heterogeneous controllers under different attack cases.

Table 1
Notations used in this paper.

Notation	Definition
V	Set of network nodes
v_i	Network node v_i
L	Set of network links
S	Set of switches
s_i	Switch s_i
λ_i	Flow request rate of switch s_i
$h(v_i, v_j)$	Packet propagation delay between node v_i and v_j
C	Set of nodes that place controllers
M	The number of controllers in the set C
c_k	The controller placed on node v_k
U	Set of types of heterogeneous controllers
$NC(u)$	Number of placed controllers with type u
$\omega(u)$	Controller capacity
α	Controller synchronization factor
Ω	Set of controllers that have been compromised by attackers
$VU(u)$	Number of known vulnerabilities of controller with type u

Table 2
Variables used in this paper.

Variable	Definition
$c_k(u)$	Binary variable, indicating whether controller with type u is placed on node v_k
t_k	The type of controller at node v_k
x_{ik}	Binary variable, indicating whether switch $v_k$connects controller on node v_k
l_{ij}	Binary variable, indicating whether there is a link between nodes v_i and v_j
D_{ik}	End-to-end delay between controller and switch
DP_{ik}	Packet propagation delay
DC_k	Controller processing delay
π	Network end-to-end delay
η_k	Controller utilization rate
CUV	Variance of the heterogeneous controller utilization rates
$PCF_k(u)$	Controller fault rate
$PCPF$	Control plane fault rate
$CF(u)$	Controller fragility coefficient
P	Probability of successful vulnerability probing
MA_k	Malicious attack coefficient
$\zeta(u)$	Attacker prior knowledge coefficient
$VU(u,u^*)$	Set of mutual vulnerabilities of heterogeneous controllers

$$l_{ij} = \begin{cases} 1 v_i directlyconnectswith v_j \\ 0 otherwise \end{cases} \tag{1}$$

Typically, controllers can be placed on any nodes in G. Different from the traditional CPP solutions, here we consider placing the heterogeneous controllers in the SDN, which means that controllers placed have multiple types. C is a set of M controller locations, and $C = \{c_k | k = \mathrm{argmax} c_k(u), \forall u, \forall v_k \in V\}$. c_k represents a node where the controller is placed. Binary variable $c_k(u)$ indicates the heterogeneous controller placement, where $c_k(u) = 1$ denotes the controller with type u is placed on node v_k, as shown in Eq. (2). $U = \{1, 2, ..., u\}$ is a set of types of heterogeneous controllers. The total number of heterogeneous controller types is $|U|$, and $NC(u)$ is the number of heterogeneous controllers with type u in the SDN. Since the heterogeneous controllers have different processing capabilities [19], $\omega(t_k)$ shows the capacity of the controller at a particular node v_k, where t_k indicates the type of controller at node v_k. One has $t_k = u$, if and only if $c_k(u) = 1$. It should be noted that controllers with the same type have equal capacities. In order to be more consistent with the real network scenario and reduce the placement costs, we set each switch to be controlled by only one master controller [7,10]. Binary variable x_{ik} is introduced to show the mapping relationship between the controller and switch managed, where $x_{ik} = 1$ denotes the switch on node v_i is managed by the controller on node v_k, as shown in Eq. (3).

$$c_k(u) = \begin{cases} 1 controllerwithtypeuisplacedonv_k \\ 0 otherwise \end{cases} \tag{2}$$

$$x_{ik} = \begin{cases} 1 switchonv_i ismanagedbycontrolleronv_k \\ 0 otherwise \end{cases} \tag{3}$$

3.2. Metrics for heterogeneous controller placement

As discussed above [8], the CPP is complex, especially for heterogeneous controllers. In this paper, for the sake of QoS and security, we consider the following metrics during formulating heterogeneous controller placement.

3.2.1. End-to-end delay

In the SDN network, end-to-end delay is one of the important criteria for evaluating QoS. The lower the end-to-end delay, the better the QoS performance. Specifically, the end-to-end delay D_{ik} between switch v_i and controller c_k defines the time experienced from a new flow arrival at switch v_i to switch v_i receiving Packet_Out message responded by the controller, as shown in Fig. 2.

The end-to-end delay generally includes four parts: 1) Switch processing delay, 2) Packet transmission delay, 3) Packet propagation delay, and 4) Controller processing delay. As hardware technology grows, high-performance switches have been widely developed (e.g., Pica8 [28] and Cisco [29]), producing negligible switch processing delays. Moreover, in a high-speed network (e.g., in a backbone network), the packet transmission delay is trivial [12,26]. Therefore, the end-to-end delay D_{ik} between switch on node v_i and controller on node v_k can be computed in Eq. (4),

$$D_{ik} = 2 \cdot DP_{ik} + DC_k \tag{4}$$

where DP_{ik} is the packet propagation delay between switch s_i and controller on node v_k, and DC_k is the processing delay of controller on node v_k. Next, we will analyze these two parameters in detail.

The packet propagation delay DP_{ik} defines the minimum packet propagation delay between switch on node v_i and controller on node v_k, as shown in Eq. (5).

$$DP_{ik} = h(v_i, v_k) \cdot x_{ik} \tag{5}$$

As shown in Fig. 2, the interactions between controller and switches can be regarded as an independent *M/M/1* queuing model [8], where the controller and the flow requests sent from switches are considered as the service counter and customers, respectively. Based on the Little Law, the average waiting time (processing delay) can be calculated as the inverse of the difference between the processing rate of the service counter and the arrival rate of the customers. As an analogy, the processing delay depends on not only the capacity of the controller itself but also the flow requests sent by the controlled switches. Besides, the synchronization cost between controllers further reduces the available capacity of the controller. In the multi-controllers SDN, a controller also communicates with the other controllers to synchronize the global network state. In order to compute the effect of state synchronization between controllers on the network performance, we consider the worst-case scenario in which each controller communicates with all remaining controllers periodically. The synchronization costs of the controller are proportional to the total number of controllers placed in the network [39,50]. Therefore, the controller processing delay DC_k is computed in Eq. (6),

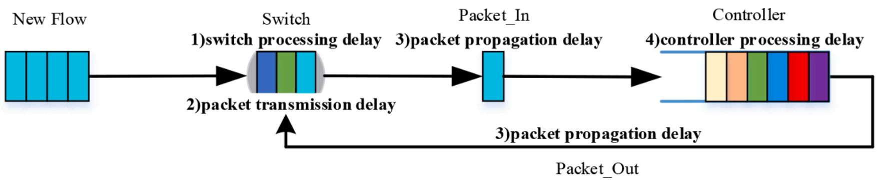
New Flow
Switch
1)switch processing delay
2)packet transmission delay
Packet_In
3)packet propagation delay
Controller
4)controller processing delay
3)packet propagation delay
Packet_Out

Fig. 2. New flow processing in SDN.

$$DC_k = \frac{1}{\omega(t_k) - \sum_{i=1}^{N} \lambda_i \cdot x_{ik} - \alpha \cdot (M-1)} \tag{6}$$

where $\omega(t_k)$ shows the capacity of the controller at a particular node v_k. λ_i is the flow request rate of switch s_i. $\alpha \cdot (M-1)$ is the synchronization cost, and α is the synchronization factor. Regardless of the types of controllers, the communication messages involved in their state synchronizations have almost no differences, including heartbeat message, flow table information, topology information, and so on. To simplify the analysis, we consider there is weak consistency between controllers, and all controllers have the same synchronization factor. Parameter M is the total number of controllers in the SDN network.

3.2.2. *Controller utilization rate*

Controller utilization rate denotes the ratio of controller loads to controller capacity, where controller loads include two parts: processing flow requests and synchronizing states. Actually, the controllers with different types have different capacities, and the controller utilization rate has a significant effect on SDN network performance. When the controller utilization rate is too high, the controller cannot process the flow requests from switches in time, resulting in packet loss, network congestion, and even controller crash. On the contrary, when the controller utilization rate is too low, the controller resources are idle and wasted. Ideally, QoS of the SDN network reaches optimal if and only if all heterogeneous controllers have the same utilization rates. Therefore, the controller utilization rate η_k is computed in Eq. (7).

$$\eta_k = \frac{\sum_{i=1}^{N} \lambda_i \cdot x_{ik} + \alpha \cdot (M-1)}{\omega(t_k)} \tag{7}$$

3.2.3. *Controller fault rate*

In this paper, we pay close attention to the controller faults caused by controller vulnerabilities, while the controller faults caused by physical failures are beyond the scope of this paper. This is because the physical failures are related to some natural factors (e.g., device aging, natural environment [7]). It should be worth noting that the controller fault rate depends on not only the security of the controller itself but also the malicious attackers. On the one hand, the security of the controller itself is related to the number of existed vulnerabilities [19]. Generally, the lower the security of the controller, the higher the controller fault rate. On the other hand, if the attackers prefer one controller that has a certain type or is placed on a critical location, this controller will be attacked more frequently and more prone to a fault [15]. Based on the above analysis and references [56,57], for the controller with type u on node v_k, its controller fault rate is represented as $PCF(t_k)$, as shown in Eq. (8),

$$PCF(t_k) = 1 - e^{-CF(u) \cdot MA_k} \tag{8}$$

where $CF(u)$ is the fragility coefficient of the heterogeneous controller with type u. If $c_k(u) = 1$, $t_k = u$. MA_k is the malicious attack coefficient. This equation indicates that the value of the controller fault rate is usually between 0 and 1 but cannot be 0 or 1. This is because no controller is safe or unsafe absolutely. Next, we will analyze the referred two parameters ($CF(u)$ and MA_k) in detail.

For the controller with type u, $CF(u)$ depends on the attackers' prior knowledge $\zeta(u)$, the number of vulnerabilities $VU(u)$, and the probability of successful vulnerability probing $P = 1 - e^{-t}$ (t is the number of probing attacks launched by the attackers) [63]. For the reason of experience [56,57], we compute $CF(u)$ with Eq. (9).

$$CF(u) = e^{\zeta(u)} \cdot VU(u) \cdot P \tag{9}$$

Although we consider placing heterogeneous controllers in the network, they may still have several homogeneities in the actual designs. For example, both Floodlight and ONOS are programmed in Java. After attacking one type of controller successfully, the attackers can utilize the prior knowledge obtained in the attack process to guide the subsequent attacks. Therefore, we compute attackers' prior knowledge $\zeta(u)$ for the controller with type u in Eq. (10),

$$\zeta(u) = \max\left(\frac{VU(u,u^*)}{VU(u^*)}\right), u^* \in \Omega \tag{10}$$

where $VU(u^*)$ is the total number of known vulnerabilities of the controller with type u^*, $VU(u,u^*)$ is the number of mutual vulnerabilities between the controllers with type u and type u^*, and Ω represents a set of controllers compromised by the malicious attackers. Specifically, $\zeta(u) \in [0,1]$.

Eq. (10) indicates that when adversaries try to attack the controller with type u, they can utilize the attack knowledge learned from controllers that have been compromised. The attackers' prior knowledge coefficient is not only positively associated with the total number of known vulnerabilities of the target controller but also negatively associated with the mutual vulnerabilities between the target controller and the controllers compromised. The more similar the target controller and the controllers compromised are, the more the mutual vulnerabilities between them. Here, referring to the Common Vulnerabilities and Exposures [59], National Vulnerability Database [60], and several methods proposed by the existing work [20,61,62], we can collect the known vulnerabilities of controllers (e.g., NOX, Ryu, Floodlight, and ONOS) and analyze the mutual vulnerabilities between them. For example, fuzzy testing [62] is a typical technology used to discover the controller vulnerability, and the southbound and the northbound interfaces of the controller are taken as the object of fuzzy-testing. Besides, OpenFlow protocol and APIs of the southbound and northbound interfaces are analyzed in detail to determine the variables of fuzzy-testing.

To simplify the analysis, we make an assumption that malicious attack coefficient MA_k is mainly related to the locations of the controller, because the attackers always regard the controllers placed on the critical nodes in the network as their attack targets to maximize the attack gains. Referring to the concept of node degree of the complex network [48], we compute MA_k in Eq. (11) and will take into account more complicated factors to optimize MA_k in the future.

$$MA_k = \frac{1}{N-1} \cdot \sum_{i=1}^{N} l_{ik} \tag{11}$$

3.3. *HeCPP formulation*

In consideration of realistic requirements from the network operators, we propose the HeCPP and define it as follows.

Definition 1: HeCPP. For a given network and set of heterogeneous controllers, we should determine the types and number of controllers and the mapping relationships between switches and controllers to minimize network end-to-end delay, balance heterogeneous controller utilization, and lower control plane fault rate. As mentioned above, it is clearly seen that HeCPP includes three optimization objectives:

1) Network end-to-end delay

Based on Eq. (4), we can obtain the end-to-end delay between one switch and one controller. For all switches and controllers in the SDN, we compute the network end-to-end delay in Eq. (12),

$$\pi = \frac{1}{M}\sum_{k}\sum_{i=1}^{N} D_{ik} \forall v_i \in V, c_k \in C \tag{12}$$

where D_{ik} is the end-to-end delay between switch on node v_i and controller on node v_k, M is the total number of controllers placed in the network.

1) Controller utilization variance (*CUV*)

Based on Eq. (7), we can get the controller utilization rate for each controller, which reflects loads of the controller. For the HeCPP, one of the objectives is placing heterogeneous controllers reasonably to balance controller loads. Therefore, we compute the controller utilization variance (*CUV*) with Eq. (13) to show whether the heterogeneous controller loads are balanced.

$$CUV = \frac{1}{M}\left(\sum_{k}\left(\eta_k - \frac{1}{M}\left(\sum_{k}\eta_k\right)\right)^2\right) \forall c_k \in C \tag{13}$$

where η_k is the controller utilization rate of the controller on node v_k, and M is the total number of controllers in the SDN.

1) Control plane fault rate (*PCPF*)

Eq. (8) presents the controller fault rate in the network, while the control plane is constructed by all controllers. Accordingly, the control plane fault rate *PCPF* is calculated in Eq. (14),

$$PCPF = \prod_{k} PCF(t_k) \forall c_k \in C \tag{14}$$

Obviously, HeCPP is a multi-objective optimization problem with multi-constraints. Based on the above analysis and three metrics, we formulate the HeCPP in Eq. (15),

$$Objective\ \min_{i,k,u} \rho_1 \cdot \pi + \rho_2 \cdot CUV + \rho_3 \cdot PCPF \tag{15}$$

$$\text{s.t.} \begin{cases} \sum_{i=1}^{N} x_{ik} \cdot \lambda_i + \alpha \cdot (M-1) \le \beta_{t_k} \cdot \omega(t_k) \forall c_k \in C \\ t_k = u \forall c_k(u) = 1 \end{cases} \tag{16}$$

$$\sum_{k} x_{ik} = 1 \forall v_i \in V, c_k \in C \tag{17}$$

$$\sum_{u \in U} c_k(u) \le 1 \forall c_k \in C \tag{18}$$

$$u \ge 2 \tag{19}$$

$$c_k(u), x_{ik}, l_{ij} \in \{0,1\} \forall v_i \in V, c_k \in C, u \in U \tag{20}$$

Eq. (15) shows that the objectives of the HeCPP are to minimize the network end-to-end delay π, controller utilization variance *CUV*, control plane fault rate *PCPF*. Based on the weighted sum way [12,55], we reformulated the multi-objective optimization problem into a single-objective programming problem for the HeCPP. Parameters ρ_1, ρ_2, ρ_3 weigh three optimization objectives, $0 \le \rho_1, \rho_2, \rho_3 \le 1$ and $\rho_1 + \rho_2 + \rho_3 = 1$. Moreover, the weights can be adjusted dynamically according to the importance of the above objectives in different network scenarios. For example, when more attention is paid to the network delay, ρ_1 has a higher weight. Eq. (16) indicates that the controller loads cannot exceed the corresponding controller capacity. Moreover, in order to withstand unexpected traffic bursts, we introduce a slack factor β_{t_k}. This slack factor depends on the value of the controller capacity. The larger the capacity, the closer the slack factor gets to 1. Through trial and error on our problem type, we elaborately determine the value of the slack factor for each type of controller to make the processing delay as equal as possible. Eq. (17) represents that each switch only has one master controller. Eq. (18) shows that only one heterogeneous controller can be placed on one node. Eq. (19) indicates that there are no less than two types of available controllers to ensure the heterogeneity of the control plane. Eq. (20) denotes the binary variables in the network model.

4. SQHCP approach

Based on the above formulation, we pay attention to the realistic network scenario, where the number of heterogeneous controllers to be placed is indeterminate and the heterogeneous controllers with diverse types have different processing capacities and security properties. Based on the objective function (Eq. (15)), we could find that the key to solving the HeCPP is to determine the reasonable types and locations of heterogeneous controllers, and mapping relationships between switches and controllers. To this end, we propose a Secure-aware and QoS-guaranteed Heterogeneous Controller Placement (SQHCP) to solve the HeCPP, which consists of two steps. In the first step, in order to enhance the security of the control plane, we compute the types and the number of required heterogeneous controllers based on dynamic planning. In the second step, for guaranteeing QoS and reducing the complexity, we combine K-means and GA to determine the locations of heterogeneous controllers and the mapping relationships between switches and controllers.

4.1. Step 1: heterogeneous controller formulation

As discussed in Section 2, the homogeneous controller common-mode fault can be avoided effectively through placing heterogeneous controllers in the SDN network. For this purpose, in order to enhance the security of the control plane, the heterogeneous controllers are formulated in this step to identify the types and the corresponding number of heterogeneous controllers to be placed. It will be easier to solve the HeCPP if the types and the number of heterogeneous controllers can be determined in advance [9,11].

Firstly, based on the network attack surface theory [49], we analyze the relationship between the types of heterogeneous controllers and the security of the control plane in Theorem 1.

Theorem 1. *The security of the control plane increases with the types of heterogeneous controllers.*

Proof. We assume there is a set of optional heterogeneous controller types U={1, 2, 3,…, u}, where each element in the set represents a type of heterogeneous controller. The set of heterogeneous controller vulnerabilities is $\{VS(1),...,VS(u)\}$, where $VS(u)$ is a set of vulnerabilities of the controller whose type is u. Attackers could launch the probing attacks on one type of heterogeneous controller at a time. We consider a more practical case that different types of heterogeneous controllers cannot realize complete heterogeneity technically [18]. This means that $VS(u) \cap VS(u^*) \neq \varnothing$ and $VS(u) \neg\subset VS(u^*)$ for any two types (u and u^*) of heterogeneous controllers. Specifically, when we place one type of controllers in the SDN (equaling to the homogeneous controllers), the attack surface that destroys the entire control plane completely is computed as $VS(1)$. When we place two types of heterogeneous controllers, the attackers are able to destroy the entire control plane if and only if mastering their mutual vulnerabilities. At this point, the attack surface that destroys the control plane completely is computed as $VS(1) \cap VS(2)$. Because of $(VS(1) \cap VS(2)) < VS(1)$, the attack surface of the control plane will shrink. Therefore, the following conclusions could be drawn. When we place u types of heterogeneous controllers, the attack surface that destroys the control plane completely is $\cap_{u \in U} VS(u)$, and $(\cap_{u \in U} VS(u)) < VS(u)$. As a result, the attack surface shrinks as the heterogeneous controller type increases, and the security of the control plane enhances accordingly. The proof of Theorem 1 ends.

Note that although the overall security potential of the control plane improves as more controller types are included, one may envisage scenarios where, above some level of heterogeneity, the inclusion of additional controller types may yield diminishing returns in the network. Based on Theorem 1, we assume that all available types of heterogeneous controllers are considered for placement in the network. Further, a more detailed study on the cost/benefit tradeoffs associated with varying degrees of heterogeneity is deferred to future work.

Therefore, to enhance the security of the control plane, each type of available heterogeneous controller is required to be placed in the network. Moreover, the total number of controllers to be placed should be minimized to reduce device overheads and placement costs. Before modeling this problem, in Theorem 2, we have first proven that the problem of identifying the number of heterogeneous controllers can be considered as a specific complete knapsack problem.

Theorem 2. *Identifying the number of heterogeneous controllers can be considered as a specific complete knapsack problem.*

Proof. Firstly, according to the literature [30], we give a formal definition of the complete knapsack problem as follows. There are GN different kinds of goods and a knapsack with a capacity of KC. For the i^{th} kind of goods $(1 \le i \le GN)$, its volume and value are GW_i and GV_i, respectively. We should select several goods so that their total volumes are less than the knapsack capacity and their total values are maximal. Note that each kind of goods can be loaded into the knapsack repeatedly. Therefore, for the complete knapsack problem, its objective function and constraint are formulated in Eq. (21) and Eq. (22),

$$\text{Objective max} \sum_{i=1}^{gn} \sum_{j \ge 1} GV_i(j) \tag{21}$$

$$\text{s.t.} \sum_{i=1}^{gn} \sum_{j \ge 1} GW_i(j) \le KC \tag{22}$$

where gn is the number of kinds of goods loaded into the knapsack and $gn \le GN$, and parameter j shows that the number of the i^{th} kind of goods loaded in the knapsack. Eq. (21) shows that the objective is to maximize the total value of goods in the knapsack. Eq. (22) indicates that the total volume of all loaded goods is less than or equal to the knapsack capacity.

To identify the number of heterogeneous controllers, we make an analogy with the special case of the complete knapsack problem. Firstly, different types of heterogeneous controllers can be regarded as different kinds of goods. The u^{th} type of heterogeneous controllers makes an analogy with the i^{th} kind of goods. The value of the heterogeneous controller (e.g., CV_u) corresponds to the value of the good (e.g., GV_i). To simplify the analysis, the values of all heterogeneous controllers are the same, and $CV_u = 1, \forall u$. Furthermore, the controller process capacity (e.g., $\omega(u)$) makes an analogy with the volume of good (GW_i). Secondly, the total traffic (e.g., the flow requests of all switches $\sum_i \lambda_i$) can be considered as the knapsack capacity (KC). Note that we will use the knapsack capacity to refer to the total flow requests later. However, different from the objective of the complete knapsack problem, our goal is to minimize the number of controllers required. In virtue of the definition of the complete knapsack problem, the problem of identifying the number of heterogeneous controllers could be described as follows. For the given u types of heterogeneous controllers, their process capacities are $\{\omega(1), \ldots, \omega(u)\}$. The total flow requests are $\sum_i \lambda_i$. The number of heterogeneous controllers with type u is computed as $NC(1), \ldots, NC(u)$. Therefore, as a special kind of complete backpack problem, our objective is to minimize the number of required controllers, as shown in Eq. (23). Since each controller has the same value, minimizing the total values of controllers is equivalent to minimizing the number of controllers.

$$\text{Objectivemin} \sum_{i=1}^{u} \sum_{j=1}^{NC(u)} CV_i(j) \tag{23}$$

$$\text{s.t.} \sum_{i=1}^{u} \sum_{j=1}^{NC(u)} \beta_u \cdot \omega(u) \ge \sum_{i=1}^{N} \lambda_i \tag{24}$$

$$1 \le NC(u) < \left\lceil \sum_{i=1}^{N} \lambda_i \Big/ (\beta_u \cdot \omega(u)) \right\rceil \tag{25}$$

where Eq. (24) shows that total controller capacity must be greater than or equal to the total flow requests; Eq. (25) presents the constraint of the number of heterogeneous controllers. The proof of Theorem 2 ends.

Since the complete knapsack problem is NP-hard [30], determining the number of heterogeneous controllers also belongs to NP-hardness. In this paper, we solve it with the help of dynamic planning [31]. Dynamic planning is based on the Behrman optimality principle, which is used to solve NP problems widely. The principle of dynamic planning is to divide the original problem into sub-problems and then solve the sub-problem by finding the iterative relationship between the original problem and the sub-problem. Finally, it achieves the result of solving the original problem. Compared with the brute force search method, the complexity of dynamic planning is linear, which improves the solving efficiency.

Based on Eq. (23), the sub-problem $F[p][q]$ represents the minimum value that can be obtained by selecting several controllers from the former p types of heterogeneous controllers and putting them into the knapsack with q remaining space. Obviously, $0 \le p \le u$ and $0 \le q \le \sum_i \lambda_i$. Referring to [30,31], the corresponding state transfer equation of $F[p][q]$ can be computed in Eq. (26),

$$F[p][q] = \min\left\{F[p-1][q], F[p-1]\left[q - k \cdot \beta_p \cdot \omega(p)\right] + k \cdot CV_p\right\}, 0 \le k \cdot \beta_p \cdot \omega(p) \le q \tag{26}$$

where $F[p-1][q - k \cdot \beta_p \cdot \omega(p)]$ shows the minimum value that can be obtained by selecting a number of controllers from the former $p - 1$ types of heterogeneous controllers and putting them into the knapsack with $q - k \cdot \beta_p \cdot \omega(p)$ remaining space. k and $\beta_p \cdot \omega(p)$ is the number and the capacity of heterogeneous controller with type p, respectively. CV_p is the value of heterogeneous controller with type p.

Based on dynamic planning, the workflow of identifying the number of heterogeneous controllers is shown in Algorithm 1. Line 1 initializes the parameters. When there are no alternative heterogeneous controllers ($p = 0$) or no flow requests ($q = 0$), the value of $F[p][q]$ is 0. Algorithm 1 contains an iterative calculation process. In the initial iteration (Lines 2–18), we get an initial estimate of the required number of controllers by considering only total flow requests as the knapsack capacity. Specifically, Lines 2–10 computes the total number of controllers to be placed in the network. Lines 11–18 counts the number of heterogeneous controllers with different types. Then, based on this result, the synchronization cost of all controllers is computed. If the sum of the total flow requests and synchronization cost is less than or equal to the sum of the capacities of all controllers (Line 19), then the algorithm outputs the results (Line 20), including the total number of controllers and the number of heterogeneous controllers for each type. Otherwise, the knapsack capacity is updated as the sum of the total flow requests and the synchronization cost (Line 22), and the algorithm returns to Line 3 and recomputes the required number of controllers until convergence is attained. Actually, this iteration process makes the required number of controllers more accurate and reliable but only increases the complexity of the algorithm slightly. We explain the reason as follows. Since the state synchronization between controllers is not the focus of this paper, we simply consider the weak consistency between controllers. Compared with the processing flow request, the synchronization cost consumes only a few controller capacities. Therefore, the discount on capacity introduced by synchronization cost is slight. The algorithm will converge quickly without too many iterations (two iterations were observed at most in our experiments).

Next, we analyze the complexity of Algorithm 1. Firstly, we will iteratively compute the value of $F[p][q]$ by traversing u types of heterogeneous controllers and knapsack capacity. Secondly, when calculating maximum $F[p][q]$, we must select k from 0 to $q/(\beta_p \cdot \omega(p))$, and this process takes time $q/(\beta_p \cdot \omega(p))$. Thirdly, during counting the number of heterogeneous controllers with different types, we need to traverse u types. The complexity of Algorithm 1 is $O(u \cdot KC \cdot \sum q/(\beta_p \cdot \omega(p)))$, where u is total types of heterogeneous controllers, $KC = \sum_i \lambda_i$ is the knapsack capacity, $1 \leq p \leq u$ and $1 \leq q \leq KC$.

Theorem 3. *Algorithm 1 can obtain the minimum number of heterogeneous controllers.*

Proof. Here, we adopt the proof by contradiction. Firstly, we assume that the optimal solution based on dynamic planning is $Solve(NC(1), \ldots, NC(u))$. The sub-problem can be described as follows: based on the knapsack capacity $\sum_i \lambda_i$, we determine the minimum M that p types of heterogeneous controllers can reach. Therefore, the corresponding solution of the sub-problem is $Solve(NC(1), \ldots, NC(p)), p \leq u$.

Next, we suppose that $Solve(NC(1), \ldots, NC(p))$ is not optimal, and there is another optimal solution $Solve(NC^*(1), \ldots, NC^*(p))$, making $Solve(NC^*(1), \ldots, NC^*(p)) > Solve(NC(1), \ldots, NC(p))$. On this basis, we conclude that $Solve(NC^*(1), \ldots, NC^*(u)) > Solve(NC(1), \ldots, NC(u))$, so $Solve(NC(1), \ldots, NC(u))$ is not optimal, which contradicts with the original assumption. Hence, Algorithm 1 could obtain the minimum number of heterogeneous controllers. The proof of Theorem 3 ends.

4.2. Step 2: heterogeneous controller placement

From Step 1, we have obtained the total number of controllers (M) and the number of heterogeneous controllers of each type $\{NC(1), \ldots, NC(u)\}$. Based on this, we solve the HeCPP to further identify the locations of controllers and the mapping relationships between the switches and controllers.

It is clearly seen that the HeCPP is a typical multi-objective optimization problem from Eq. (15), and the objectives include minimizing network end-to-end delay, controller utilization variance, and control plane fault rate. Obviously, HeCPP is a specific type of CPP. A lot of researchers have proved the CPP is NP-hard [9,10], so the HeCPP is also NP-hard. Because of high complexity and time consumption, it's hard to meet the requirements of solving HeCPP with traditional solutions (e.g., Newton method, gradient descent method) and tools (CPLEX, Lingo).

Therefore, in order to solve the HeCPP efficiently, we combine the clustering algorithm and genetic algorithm (GA) and then introduce the heterogeneous controller placement based on improved heuristic algorithm. In line with the "divide and conquer" strategy, we first partition the entire network into M subnets, where M is the number of controllers obtained in Section 4.1. Then, we improve GA to solve the HeCPP of each subnet. Finally, the HeCPP of the whole SDN network is well solved. The benefit of our proposal is elaborated as follows. For a network with N nodes, the complexity of the search space is computed as $O(2^N)$. After the SDN network is partitioned into several subnets based on the clustering algorithm, each subnet owns N/M nodes averagely, and the complexity of the search space can be reduced to $O(M \cdot 2^{N/M})$. Next, we will introduce our proposal in detail below.

4.2.1. Network partition based on K-means clustering

Network partitioning is applied in large-scale networks in general, while the clustering algorithm is a common network partitioning method. Now, lots of researchers have proposed applying the clustering algorithm for the SDN [8,34]. Inspired by the existing work, we adopt K-means clustering to partition the SDN into M subnets, as shown in Algorithm 2. Algorithm 2 takes the packet propagation delays as the criteria to implement clustering for switch nodes in the network, where the packet propagation delays are computed by the Dijkstra algorithm. Note that Algorithm 2 only determines the number of subnets and does not involve the controller placements (e.g., which type of heterogeneous controller is deployed on which subnet). In detail, we describe the process of Algorithm 2 as follows. Line 1 computes the packet propagation delay between any two nodes. Lines 2–3 randomly selects M nodes from set V as the centroids. Lines 4–18 show the iteration process of Algorithm 2. Line 5 initializes M empty sets. From Lines 6–9, for each node in the network, we compute the distance between the node and each centroid and record the node and the centroid corresponding to maximum distance. After that, this node is added into the set corresponding to the centroid (Line 10). Lines 13–16 update the centroid. Specifically, Line 14 computes the number k^* and updates cv_{k^*} as a new centroid. The algorithm runs until the maximum number of iterations is reached, and the network is partitioned into M subnets. In order to improve the robustness of our scheme, we design a simple and efficient detection process to eliminate possible overloads in the subnets. After obtaining the result of the network partition based on the clustering algorithm, we compute the total number of flow requests for the switches in each subnet in Line 19. Then, we check to see if there is an overload subnet. Specifically, if the total number of flow requests for one subnet exceeds the maximum capacity of the available controller (Line 20), we consider that this subnet is overloaded. Further, the switch node with the smallest flow request is moved from the overloaded subnet to its neighbor subnets with the lowest traffic requirement (Lines 21–23). This detection process is repeated until there is no overloaded subnet in the network, and we update the subnets in the end (Line 24). The complexity of Algorithm 2 is $O(M^2 \cdot N \cdot I_{\max})$, where M is the number of subnets, N is the number of nodes in the network, and $I_{\max}$ is the maximum number of iterations. Moreover, the effectiveness of K-means clustering has been demonstrated widely in [36].

4.2.2. Heterogeneous controller placement based on improved GA

After the network has been partitioned into M subnets, we improve the genetic algorithm (GA) to solve the HeCPP for each subnet. In recent years, evolutionary computation has been applied to solve all kinds of NP problems, including the CPP [32,33]. Among all of the evolutionary computation algorithms, we prefer genetic algorithm (GA) for two reasons: 1) Plenty of previous researchers proven that GA can be used to solve the multi-objective optimization problems in the controller placement; 2) The solution to HeCPP in the form of the binary array is easily represented by chromosomes of GA. However, it is impractical to directly adopt GA to solve HeCPP in the large-scale network. This is because it will require to enlarge the population sizes and iteration times of GA, greatly increasing the runtime of the algorithm.

Based on [33], we describe the implementation of GA in Fig. 3, and the key elements include encoding, computing fitness value, genetic operation (selection, crossover, mutation). Moreover, in order to find the approximately optimal solution to place controllers, we set the following three heuristic rules for the HeCPP: 1) Each subnet $G_k = (V_k, L_k)$ only places one heterogeneous controller, but the heterogeneous controller can be placed on multiple subnets if necessary; 2) The controller should be preferentially placed in the node with the highest node degrees to facilitate management; 3) The subnet that has plenty of

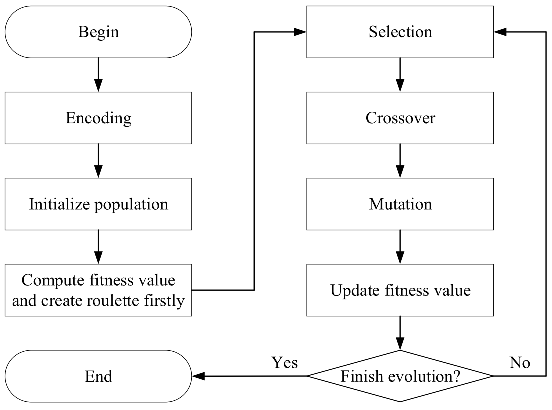

Fig. 3. The implementation of genetic algorithm.

flows should place the heterogeneous controller with low fragility coefficient and high capacity. Specifically, the first heuristic rule is reflected in the mathematical model. The last two heuristic rules are reflected in the encoding scheme of the heterogeneous controller placement.

1) Encoding

In this paper, we adopt binary encoding because the binary is easy to encode and decode, and the corresponding crossover and mutation can be realized by bit operation. Note that the improved GA is applied to each subnet. For each subnet, regardless of the location and type of heterogeneous controller, this heterogeneous controller still manages all switches of the subnet. Formally, $x_{ik} = 1, \forall v_i \in V_k, v_k \in C$, where v_i is the node in V_k and v_k is the location of heterogeneous controller. Based on the above, we just use the improved GA to determine the location and type of the heterogeneous controller for each subnet. Fig. 4 shows the binary encoding of the heterogeneous controller placement, which also represents an individual in the population of GA, where the green part and the blue part shows the location encoding and the type encoding of the heterogeneous controller, respectively. On the one hand, each bit in the location coding presents a node in the subnet. Considering the mentioned heuristic rule (2), the bit corresponding to the node that has the highest degree is more likely set to 1 in the location coding. On the other hand, each bit in the type coding shows a certain type of heterogeneous controller. Considering the mentioned heuristic rule (3), the bit corresponding to the heterogeneous controller with low fragility coefficient and high capacity is more likely set to 1 in the type coding.

1) Fitness function and constraints

The fitness function is used to determine the solution quality, usually the objective function. Therefore, for the subnet $G_k = (V_k, L_k)$, we refer to Eq. (15) and Eq. (16)–(20) to calculate its fitness function and constraints, respectively. The difference is that the domain is restricted to subnet G_k.

It must be clearly seen that these constraints have significant effects on the solution space of GA. Next, we will introduce how does GA enforce the constraints 16–20 in detail. Currently, the researchers [64] have proposed two main ways, including discarding infeasible individuals and penalty function, to deal with the constraints for GA. Specifically, the first method always checks all individuals of the population generated and directly discards those individuals that do not satisfy any of the constraints. It is the simplest and most widely used method. The second method designs a penalty function, which transforms the constrained optimization problem into the unconstrained optimization problem by reformulating the fitness function based on the objective function and the corresponding constraints. However, for the penalty function, the penalty coefficient greatly affects the quality of the solution and is difficult to select the appropriate one in practice. Based on the above analysis, we select the method of discarding infeasible individuals and improve this method to enforces the constraints for the GA. In our proposal, after the population has been initialized or updated (e.g., selection, crossover, and mutation), we check each individual in the population to see if they meet all the constraints and discard the infeasible individuals that do not satisfy any of the constraints. If the number of remaining individuals is less than the threshold of population size, we reinitialize the discarded individuals and add them back into the population. We will keep checking individuals in the population until they meet not only the constraints but also the threshold of population size.

1) Genetic operation

The genetic operations consist of selection, crossover, and mutation. In terms of the selection operation, we adopt the elite selection way. At first, the selection operation is performed according to roulette. Then, in the current population, the placement with the highest fitness value is completely copied into the next generation population. In terms of the crossover operation, we adopt the uniform crossover, as shown in Fig. 5. This means that the bits of two placement ways are exchanged with the same crossover probability, thus forming two new placement ways. In terms of the mutation operation, we adopt the bit mutation, as shown in Fig. 6. This implicates that one or more bits of the placement way are changed to the other values with the mutation probability. Moreover, during the genetic operation in the GA, in order to reflect heuristic rules 2 and 3, we modify the operations of crossover and mutation. Concretely, no matter for crossover or mutation operations, the bit corresponding to the node with the highest degree is first to be reserved as 1 in the location coding (this reflects heuristic rule 2); while the bit corresponding to the controller with low fragility coefficient and high capacity is first to be reserved as 1 in the type coding (this reflects heuristic rule 3).

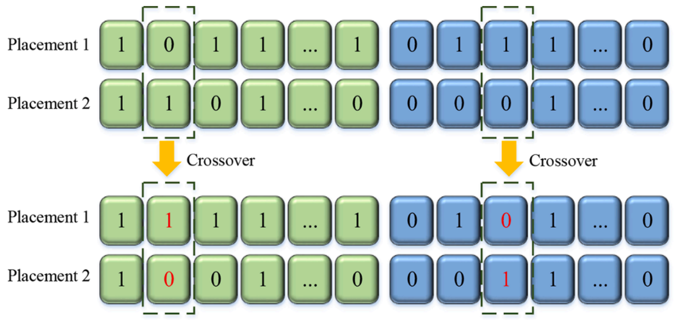

Fig. 5. Uniform crossover operation.

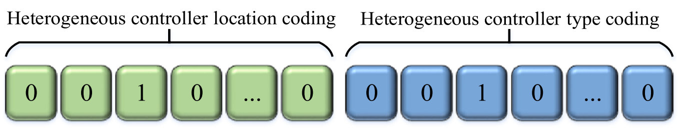
Heterogeneous controller location coding
Heterogeneous controller type coding
0 0 1 0 ... 0
0 0 1 0 ... 0

Fig. 4. Binary encoding scheme of the heterogeneous controller placement.

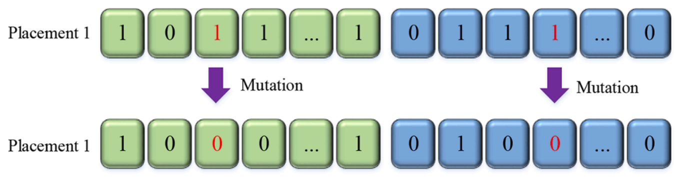

Fig. 6. Bit mutation operation.

Based on the above analysis, the heterogeneous controller placement based on improved GA is shown in Algorithm 3. The Algorithm 3 treats the subnets one after another. For better placement results, we optimize the processing order of the subnet. Specially, we consider the subnets with more flow requests should be processed first, which can make sure that a potentially wider range of controller types be available for use in them (Line 1–2). Line 4–24 implements the improved GA, including initialization, comparison of fitness values, selection, crossover, and mutation, for each subnet obtained from Algorithm 2. Note that Lines 6 and 13 revises individuals of populations that are initialized and updated to make them meet not only the constraints 16–20 but also the threshold of population size, respectively. After processing one subnet, we need to update the number and types of controllers available in set C (Line 25). During controller placement, once controllers of a particular type are exhausted, they are no longer available for use in subsequent steps. In this case, we consider that the bit corresponding to this type of controller can only be set as 0 in the type coding. Finally, we can identify the type and the location of a heterogeneous controller to be placed in each subnet. The complexity of Algorithm 3 is mainly attributed to genetic operations. The maximum number of nodes contained in each subnet is N, and the evolution iterates NE times. We consider that each population includes NI individuals at least. Therefore, the complexity of computing each subnet is $O(NE\cdot\log N\cdot NI)$. Furthermore, the genetic process requires to poll M ($0<M<N$) subnets. As a result, the complexity of Algorithm 3 is $O(NE\cdot\log N\cdot NI\cdot N)$.

Theorem 4. *The asymptotic optimality of the controller placement is true only with respect to the individual subnet-specific optimization problems.*

Proof. It should be noted that heterogeneous controller placement is based on the result of network partition in this paper. For the entire network, since Algorithm 2 has restricted the optimization domain, the combined result of Algorithm 2 and 3 can be only suboptimal, compared to an algorithm that would optimize the HeCPP problem on the entire non-partitioned network. For the subnet, we will perform the following theoretical analysis to show the optimization performance of Algorithm 3.

When Algorithm 3 calls the genetic algorithm (GA) for subnet G_k, it will record the placement that has the maximum fitness value and implements the genetic operations (e.g., selection, crossover, mutation) for this placement to get the next generation. We assume that $Q(t) = \{C_1(t), C_2(t), \ldots, C_n(t)\}$ is a population in the t^{th} generation in the GA, where $C_n(t)$ is an individual (namely a specific heterogeneous controller placement scheme) in $Q(t)$. $Z(t) = \max\{F_k(C_x(t))|x = 1, 2, \ldots, n\}$ shows the maximum fitness value in $Q(t)$, where the function $F_k(\cdot)$ is the fitness function of subnet $G_k = (V_k, L_k)$ shown in Eq. (15). Moreover, the global maximum fitness value is computed as $Z^* = \max\{F_k(C)|C \in Q(t), t > 0\}$, where C is one individual (namely, a set of nodes that place controllers) in all generations.

Actually, in Algorithm 3, the populations evolve from one generation to the next. So, the population evolution can be considered as a stochastic process, and we analyze it with the Markov chain [58]. In terms of one population $Q(t)$, it may include several states, whose set Γ can be divided into two parts: state set Γ_o with the optimal individual and state set Γ_{no} without the optimal individual. The transition probability of one population from states $Q_i(t) \in \Gamma$ to $Q_j(t) \in \Gamma$ is r_{ij}. Moreover, the state transition matrix can be expressed as $R = \{r_{ij}\}$. Therefore, in the t^{th} generation, the probability $P_j(t)$ that the population is in state j can be computed in Eq. (27),

$$P_j(t) = \sum_{Q_i(t)\in\Gamma} P_i(0) r_{ij}(t) t > 0 \tag{27}$$

where $P_i(0)$ is the probability that the population is in state i in the initial generation and $r_{ij}(t)$ is the transition probability of one population from state i to state j in the t^{th} generation.

In the Algorithm 3, each generation of the population is only related to the previous generation and independent of the initial generation. So, the evolution of Algorithm 3 can be regarded as a homogeneous Markov process [58]. Based on this, the stable probability distribution of $P_j(t)$ is independent of its initial probability distribution, as shown in Eq. (28).

$$P_j(\infty) = \sum_{Q_i(\infty)\in\Gamma} P_i(\infty) r_{ij} > 0 \tag{28}$$

For the genetic operation in Algorithm 3, we set that the controller placement scheme with the highest fitness value is completely copied into the next generation population. Therefore, we can derive that $Q_j(t) \in \Gamma_o$ if and only if there is $Q_i(t) \in \Gamma_o$. Since state i can be initialized any values from the state set Γ, it follows Eq. (29),

$$\lim_{t\to\infty} P(Z_i(t) > Z_i(0)) = 1 \tag{29}$$

where $Z_i(0)$ and $Z_i(t)$ are the maximum fitness values of population $Q_i(t)$ in the initial generation and the t^{th} generation, respectively.

Therefore, based on Eq. (29), the probability that $Q_i(t) \in \Gamma_o$ is greater than 0. Further, we can deduce the probability that $Q_i(t) \in \Gamma_o$ is greater than 0 and obtain the following result, as shown in Eq. (30).

$$\lim_{t\to\infty} P\{Z_j(t) = Z^* > Z_i(t)\} = 1 \tag{30}$$

When the number of generations (t) approaches infinity, Eq. (30) presents the probability is 1 if the maximum fitness values $Z_j(t)$ of one population $Q_j(t)$ is equal to the global maximum fitness value Z^*. Thus, based on the above analysis, for one subnet G_k, after the genetic operation and population evolution, Algorithm 3 can converge to the approximately optimal heterogeneous controller placement scheme in the finite generations (t) in the end. The proof of Theorem 4 ends.

5. Simulation results

5.1. Simulation environment

(1) Simulation platform

In this paper, we adopt one physical server (Intel E3–1230, 4 Cores, 2.3 GHz, 32 GB RAM, Ubuntu 16.04) as the simulation platform. We also select four SDN controllers as available placement targets: NOX, Ryu, Floodlight, and ONOS.

(1) Topology environment

We implement our experiments on the existing topologies, all of which are from Internet Topology Zoo [35]. Internet topology zoo is a publicly available topology set, and a lot of researchers have used it to study the CPP. To make our experiment more representative, we select three topologies with different sizes (based on the number of nodes N): OS3E (small-scale topology, $N < 50$), Columbus (medium-scale topology, $50 \le N < 100$), and RF-II (large-scale topology, $N \ge 100$), as shown in Table 3. We compute the hops between any two nodes in virtue of the Dijkstras algorithm. At the beginning of the simulation, there are no packets in the network. After the simulation starts, each switch generates flow requests randomly and the simulation runs for five minutes. We have observed that the throughput and delay reach a steady state after 20 s.

Table 3
Experimental topologies.

	OS3E	Columbus	RF-II
Nodes	34	70	108
Links	42	85	306

(1) Parameters setting

In order to simulate the real traffic, we hold that the flow request rate λ of each switch follows the Poisson distribution, and each switch will be assigned a rate value independently. By this way, we not only guarantee the asymmetry of network traffic but also make our simulation closer to the real network environment. Based on the following literatures [15,19,20] and the empirical reason, the parameters (controller capacity ω, the number of vulnerabilities VU, prior knowledge coefficient ζ) for different controllers are set as NOX (80, 32, 0.1), Ryu (130, 36, 0.1), Floodlight (190, 55, 0.3), ONOS (300, 67, 0.4), respectively. Since the state synchronization between controllers is not the focus of this paper, we simply consider the weak consistency between controllers. The value of the synchronization factor is determined based on empirical reason, and α is set to 0.1 following existing work [50]. By trial and error on our problem type, the slack factor β is set to 0.85, 0.90, 0.95, and 1 for NOX, Ryu, Floodlight, and ONOS, respectively. During the simulation, we set different parameters for the proposed algorithms. On the premise of not affecting the experimental results, the threshold of the population size of GA is set to be 20 times the number of nodes, the maximum evolution times are 1000, and the crossover and mutation probabilities are 0.3 and 0.2, respectively. To eliminate random error, all simulations run 100 times.

5.2. Results evaluation

In this subsection, we evaluate the performance of SQHCP approach from the perspectives of security and QoS. In terms of security, we evaluate SQHCP approach for enhancing the security of the control plane. As far as we know, we are the first to propose the heterogeneous controller placement. We vary the types of heterogeneous controllers placed to evaluate the performance of SQHCP approach. In terms of the QoS, we will compare SQHCP approach with two representative CPP solutions K-Center [9] and DBCP [43].

- K-Center clusters the nodes based on propagation delay in the network and places the controllers in the clustering centers.
- DBCP is a density-based controller placement approach. It also divides the network into several subnets based on the density of switches and places that controller as closely as to other subnets.

As previously mentioned, we first propose the heterogeneous controller placement. Therefore, in order to make a comparison with our proposal, we simply refine the existing approaches (K-Center and DBCP) as follows.

Firstly, we have introduced the K-Center algorithm and its modification in detail. K-Center is a well-known algorithm first applied to solve the controller placement problem (CPP), and please see literature [9] for its detailed explanation. The original K-Center algorithm only considers homogeneous controllers. In order to compare with our proposal, we simply modify the K-Center algorithm and make it suitable for a network with heterogeneous controllers. The workflow of the modified K-Center is described in Fig. 7, where the solid line area and the dashed line area represents the original K-Center and our modification for K-Center, respectively. In step 1, for a given SDN network, we compute the propagation delay between any two nodes. In step 2, we implement k-center clustering for all nodes to minimize the worst-case delay, which is defined as the maximum node-to-controller propagation delay. After that, we can obtain several clusters (a cluster is equivalent to a subnet),

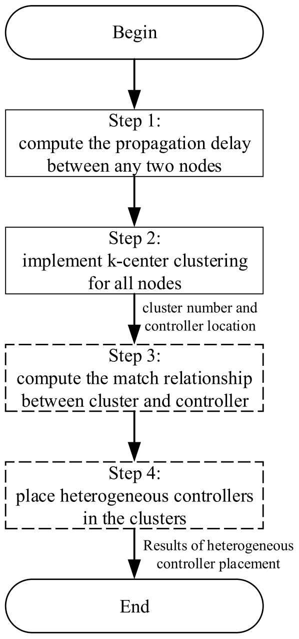
Begin
Step 1:
compute the propagation delay between any two nodes
Step 2:
implement k-center clustering for all nodes
cluster number and controller location
Step 3:
compute the match relationship between cluster and controller
Step 4:
place heterogeneous controllers in the clusters
Results of heterogeneous controller placement
End

Fig. 7. The workflow of modified K-Center.

and the controller location is on the clustering center, and each location only has one controller. Please note that K-Center only considers propagation delay, so these clusters may vary in size (namely, different clusters have different number of switches). In step 3, we will rank the clusters in descending order according to their sizes. The first cluster will definitely select the controller with maximum capacity (e.g., ONOS). Based on the ratio of the capacity of the ONOS controller to the size of the first cluster, we in turn deduce the cluster size that the other controllers (e.g., NOX, Ryu, Floodlight) can match. In step 4, the cluster will select the specific heterogeneous controller that approximately matches its size. More importantly, some clusters may be adjusted adaptively (e.g., moving few nodes to neighbor clusters) so that all types of heterogeneous controllers can match them well. Finally, the modified K-Center algorithm outputs the results of heterogeneous controller placement.

Secondly, we have also introduced the DBCP algorithm and its modification in detail. DBCP is a new placement approach named Density Based Controller Placement, please see literature [43] for its detailed explanation. The original DBCP algorithm also only considers homogeneous controllers. In order to compare with our proposal, we simply modify the DBCP algorithm and make it suitable for a network with heterogeneous controllers. The workflow of modified DBCP algorithm is described in Fig. 8, where the solid line area and the dashed line area represents the original DBCP and our modification for DBCP, respectively. In step 1, based on the distance between switches, we compute two quantities for each switch in the network: its local density and its distance to switches with higher density. In step 2, by computing the minimum distance between the switch and any other switch with higher density, we implement a density-based switch clustering algorithm, and the cluster centers are the switches whose minimum distance values are anomalously large. After the cluster centers are obtained, each remaining node is assigned to the same cluster as its nearest neighbor with higher density. The number of clusters is equal to the number of subnets, and one controller can only be placed per cluster. In step 3, we assign the available heterogeneous controllers to each cluster. This procedure is similar to the combination of steps 3 and 4 of the modified K-Center. However, different from the modified K-Center, the modified DBCP considers the total load of switches rather than the size of the cluster. Namely, we identify the match relationship between the controller and cluster based on the ratio of the capacity of the controller to the total load of switches of the cluster. In step 4, the placement

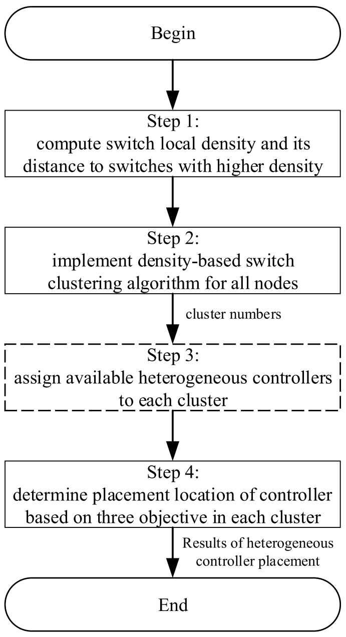
Begin
Step 1:
compute switch local density and its distance to switches with higher density
Step 2:
implement density-based switch clustering algorithm for all nodes
cluster numbers
Step 3:
assign available heterogeneous controllers to each cluster
Step 4:
determine placement location of controller based on three objective in each cluster
Results of heterogeneous controller placement
End

Fig. 8. The workflow of modified DBCP.

location of the controller can be determined based on three objective functions (e.g., controller-to-switch delay, inter-controller delay, and network reliability) in each cluster. We can either find a best placement or a set of Pareto-optimal placements by using a traversal method within each subnet.

It should be noted that the modifications we made to the existing schemes (K-Center and DBCP) do not affect their core mechanisms. The modifications will allow them to place heterogeneous controllers for easy comparison with our proposal. For simplicity, we still use the names of K-Center and DBCP to refer to the modified schemes.

5.2.1. *The security of the control plane*

In this experiment, we evaluate the performance of SQHCP on the security of the control plane. Specifically, referring to Eq. (14), we use the control plane fault rate to represent the security of the control plane. Firstly, to show the performance of SQHCP under different heterogeneous controller situation, we list all possible combinations of heterogeneous controllers in Table 4, where U is a set of types of heterogeneous controllers. The available controllers are NOX, Ryu, Floodlight, and ONOS. When we place one type of heterogeneous controller ($|U| = 1$), based on the combination formulas in mathematics, $C_4^1 = 4$ and there are four cases: NOX, Ryu, Floodlight, and ONOS. Similarly, $|U| = 2$, then $C_4^2 = 6$; $|U| = 3$, then $C_4^3 = 4$; $|U| = 4$, then $C_4^4 = 1$. The number of probing attacks for one controller is set to 100, and we compute the control plane fault rate under 15 cases ($C_4^1 + C_4^2 + C_4^3 + C_4^4 = 15$). Since this paper first places heterogeneous controllers in the SDN, we conduct a self-comparison of SQHCP.

Figs. 9–11 shows the control plane fault rate with different case numbers under three topologies. X axis is the case number in Table 4, where cases 1 to 4, cases 5–10, cases 11–14, and case 15 show placing one, two, three, and four types of heterogeneous controllers in the network, respectively.

From OS3E to RF-II, as the topology nodes increase, the control plane fault rate also raises. This is because increased nodes expand the attack surface. However, for the same topology, the more types of heterogeneous controllers in the network, the lower the control plane fault rate. Besides, the combinations of heterogeneous controllers also affect the control plane fault rate. For example, in Figs. 9–11, case 5 (NOX+Ryu) produces a lower control plane fault rate than case 10 (Floodlight+ONOS). This indicates the number of vulnerabilities and the prior knowledge coefficient of the heterogeneous controller should be considered in placing controllers.

Based on the above results, we evaluate how the performance of SQHCP varies with the number of probing attacks. We select the optimal combinations (case 1 for $|U|$=1, case 5 for $|U|$=2, case 11 for $|U|$=3, and case 15 for $|U|$=4) under different types of heterogeneous controllers from Figs. 9–11 and compute the Cumulative Distribution Function (CDF) of control plane fault rate by changing the number of probing attacks. We suppose that the probability of successful vulnerability probing is $P = 1 - e^{-t}$, where t is the number of probing attacks launched by the attackers [63]. Attackers can exploit the vulnerability to attack the specific heterogeneous controller after successfully probing it. The simulation results are shown in Figs. 12–14.

Regardless of any topologies, the types of heterogeneous controllers significantly affect the control plane fault rate. When $|U|$=1, all controllers have the same type, which is equivalent to the homogeneous. In this case, the attackers destroy the entire control plane with fewer probing attacks. As $|U|$ increases, the control plane fault rate falls and the attackers must conduct more probing attacks. The heterogeneous controller placement has eliminated the homogeneous controller common-mode fault and increased the difficulty of probing attacks. Therefore, through placing heterogeneous controllers, SQHCP degrades the control plane faults and enhances its security effectively.

Finally, we also record the controller synchronization costs introduced by heterogeneous controllers. We assume that controllers synchronize the states between themselves via pair-wise communications

Table 4
All combinations of heterogeneous controllers.

Types of heterogeneous controllers	Case number	Combinations of heterogeneous controllers
$\|U\| = 1$	1	NOX
	2	Ryu
	3	Floodlight
	4	ONOS
$\|U\| = 2$	5	NOX+Ryu
	6	NOX+Floodlight
	7	NOX+ONOS
	8	Ryu+Floodlight
	9	Ryu+ONOS
	10	Floodlight+ONOS
$\|U\| = 3$	11	NOX+Ryu+Floodlight
	12	NOX+Ryu+ONOS
	13	NOX+Floodlight+ONOS
	14	Ryu+Floodlight+ONOS
$\|U\| = 4$	15	NOX+Ryu+Floodlight+ONOS

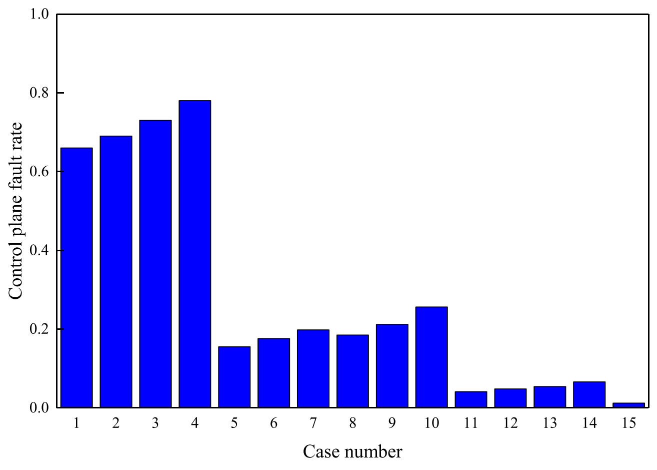

Fig. 9. Control plane fault rate in OS3E topology.

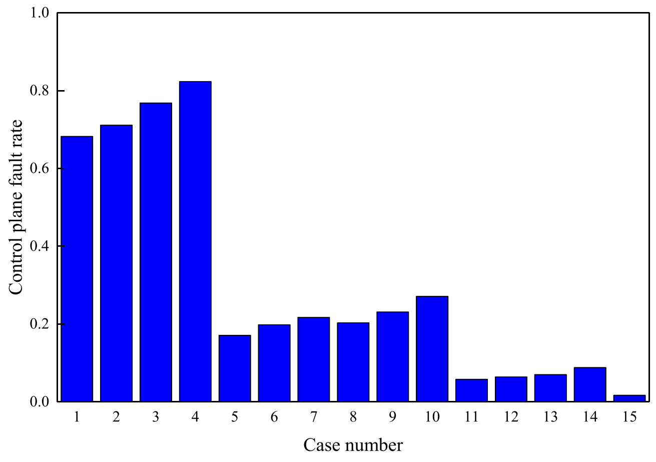

Fig. 10. Control plane fault rate in Columbus topology.

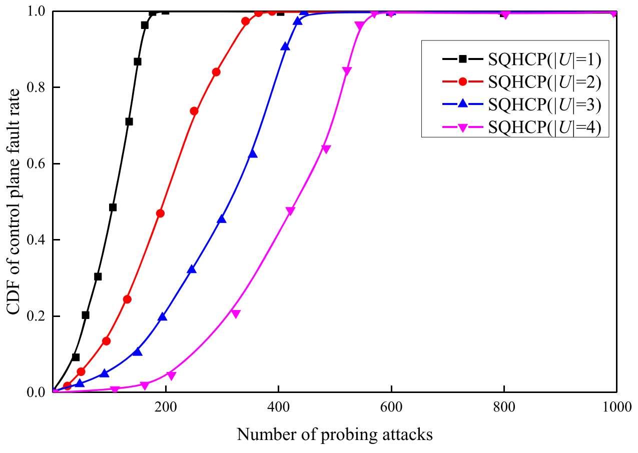

Fig. 13. CDF of control plane fault rate in Columbus topology.

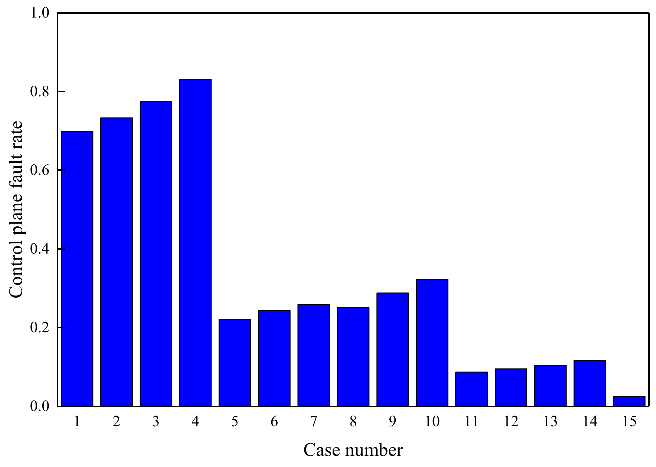

Fig. 11. Control plane fault rate in RF-II topology.

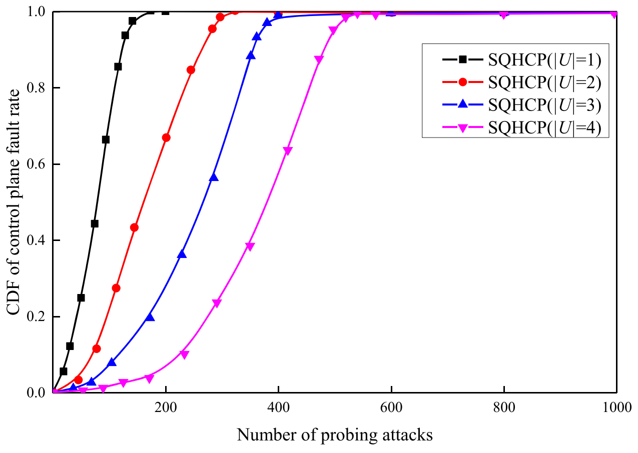

Fig. 14. CDF of control plane fault rate in RF-II topology.

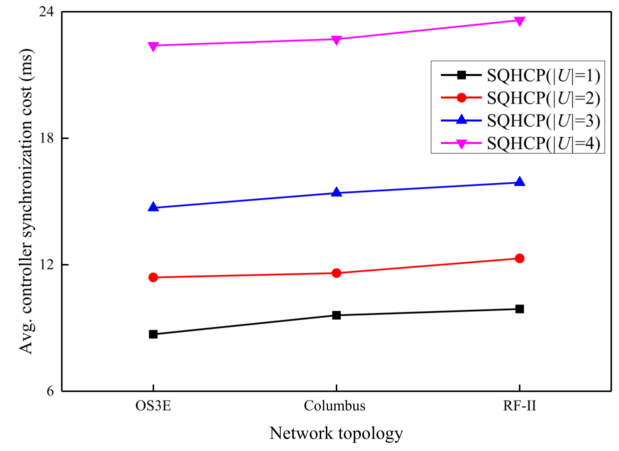

Fig. 15. Effect of heterogeneous controllers on synchronization cost.

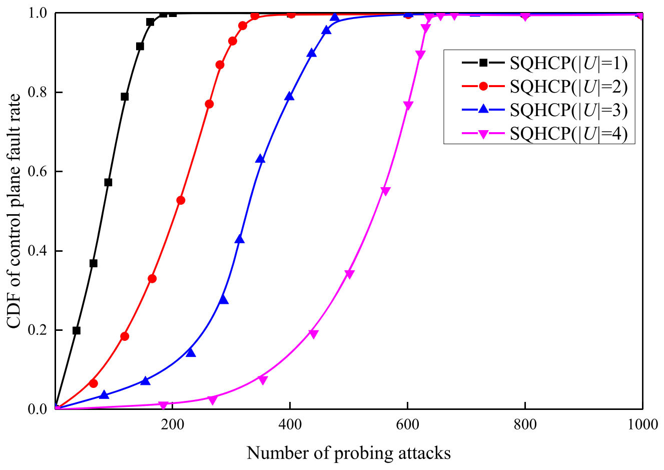

Fig. 12. CDF of control plane fault rate in OS3E topology.

between each controller pair. Fig. 15 shows the result. As the network topology scale enlarges, the average controller synchronization costs increase slightly. However, in the same topology, the types of heterogeneous controllers obviously affect the controller synchronization costs. For $|U|=1$, there are homogeneous controllers in the network, and the controller synchronization cost is the lowest. Compared to this, the heterogeneous controllers ($|U|>1$) have produced more synchronization costs. The bigger $|U|$, the higher costs. Because the heterogeneous controllers require to complete the communication processes by vertical handoff before state synchronization. Although the heterogeneous controllers introduce more controller synchronization costs, we contend that this tradeoff between cost and security is acceptable in the SDN network since the heterogeneous controller placement will not only provide stable network services but also greater confidence in security. Therefore, in subsequent simulations, without special instructions, we place all available heterogeneous controllers by default, that is, $|U| = 4$.

5.2.2. *Total number of controllers*

Based on the topologies in Table 3, we compare the total number of controllers required by three approaches. K-Center and DBCP also place heterogeneous controllers ($|U| = 4$).

The results are shown in Fig. 16. The total number of controllers obtained from three approaches are almost identical under the small-scale topology. Moreover, Table 5 also indicates how many controllers of each particular type are used for different approaches. As the topology size expands, the controllers increase gradually. From OS3E (34 nodes) to RF-II (108 nodes), K-Center requires the most controllers (from 6 to 21 controllers), which is almost proportional to the increment of nodes. DBCP determines the number of controllers based on the density of the network and performs well in terms of small and medium topologies. Compared to K-Center and DBCP, SQHCP requires the least controllers. The following reasons can explain these results. SQHCP considers the heterogeneous controller placement as a complete knapsack problem and solves it based on dynamic planning. Meanwhile, according to the network characteristics, it places the controllers through improving the heuristic algorithms. Therefore, SQHCP effectively obtains the minimum number of controllers that meet the network requirements, which is reduced by 21.8% averagely compared to the other approaches.

5.2.3. *Network end-to-end delay*

In this experiment, we evaluate the performance of SQHCP from the perspective of the end-to-end delay. To make simulation fair, three approaches place all available heterogeneous controllers ($|U| = 4$). Figs. 17–19 shows the cumulative distribution functions (CDF) of the maximum end-to-end delays of three topologies, where the results of K-Center, DBCP, and SQHCP are shown in blue, black, and red lines, respectively. It is observed that K-Center and SQHCP have a narrow gap but are superior to DBCP under small-scale topology OS3E. DBCP places the controller in the boundary node of the subnet, so there are long delays between the center nodes and controller. As the topology enlarges, the hops between nodes also increase. K-Center only considers the distance factor and cannot ensure the optimal controller placement. Therefore, in both Columbus and RF-II topologies, the delay gaps between K-Center and SQHCP are bigger.

Fig. 20 further quantifies the end-to-end delay results and describes the average end-to-end delays of the three approaches. For all topologies, SQHCP has reduced the end-to-end latency by 15.7% averagely compared with K-Center and DBCP.

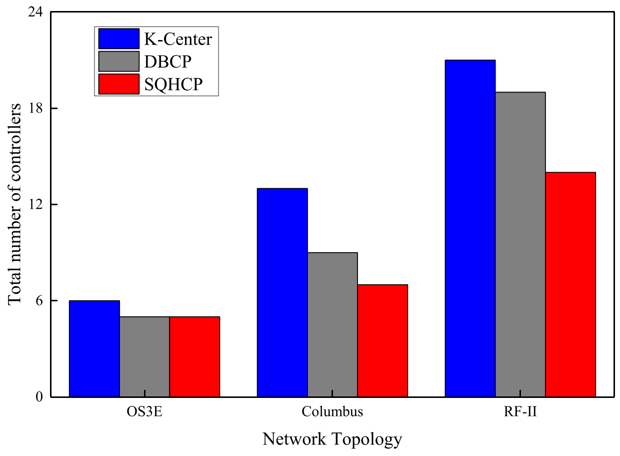

Fig. 16. The total number of controllers in different topologies.

Table 5
The information of heterogeneous controller quantity.

Topology	Controller	K-Center	DBCP	SQHCP
OS3E	NOX	1	1	1
	Ryu	2	1	2
	Floodlight	1	1	1
	ONOS	2	2	1
Columbus	NOX	2	2	1
	Ryu	3	2	1
	Floodlight	2	1	2
	ONOS	6	4	3
RF-II	NOX	2	3	2
	Ryu	4	3	3
	Floodlight	3	4	4
	ONOS	12	9	5

Algorithm 1
Heterogeneous controller formulation based on dynamic planning.

```
Input Total types of heterogeneous controllers$u$
    Heterogeneous controller capacities $\{\omega(1),\omega(2),...,\omega(u)\}$
    Total flow requests $\sum_{i=1}^{N}\lambda_i$
Output Total number of controllers $M$
    Number of heterogeneous controllers $\{NC(1),NC(2),...,NC(u)\}$
1   initialization $F[0][q]\leftarrow 0$, $F[p][0]\leftarrow 0$
2   knapsack capacity $KC = \sum_i \lambda_i$
3   for $p = 1$ to $u$
4     do for $q = 1$ to $KC$
5       do for $k = 0$ to $q/(\beta_p\cdot\omega(p))$
6         if $q \geq k\cdot\beta_p\cdot\omega(p)$
7         $F[p][k]\leftarrow\max(F[p][k], F[p-1][q-k\cdot\beta_p\cdot\omega(p)] + k\cdot CV_p)$
8       endif
9   endfor
10   return $F[u][KC]$ and $M$
11   $p\leftarrow u$, $q\leftarrow KC$, $NC(p) = 0$
12   while $(p > 0 \& q > 0)$
13     do if $(F[p][q] = F[p][q-\beta_p\cdot\omega(p)] + 1)$
14       then $NC(p) = NC(p) + 1$
15       $q\leftarrow q - \beta_p\cdot\omega(p)$
16     else $p\leftarrow p - 1$
17   endif
18   endwhile
19   if $KC + \alpha\cdot M\cdot(M-1) \leq \sum_u NC(u)\cdot\beta_u\cdot\omega(u)$
20     output $M = \sum_u NC(u)$ and $NC(u)$ for each $u$
21   elseif
22     $KC = KC + \alpha\cdot M\cdot(M-1)$ and return Line 3
23 endif
```

5.2.4. *Controller load balance*

In this experiment, we compute the standard deviation of controller utilization rate (standard deviation for short), which is the square root of *CUV* (*CUV* is shown in Eq. (13)), so as to evaluate the performance of the heterogeneous controller load balance. The smaller the standard deviation, the more balanced loads of heterogeneous controllers. Similar to the above experiments, we suppose that K-Center, DBCP, and SQHCP place four types of heterogeneous controllers ($|U| = 4$).

We obverse the performances of the three approaches by increasing the flow request rates of the switches gradually. Specifically, the flow request rates are divided into five intervals from 0 to 100 kreq/s. Each switch is independently assigned a flow request rate value, drawn uniformly from the specific interval. By changing the flow request rates of the switches, Figs. 21–23 shows the standard deviations under three topologies, respectively. On the one hand, the larger the topology scale, the greater the standard deviation; on the other hand, the more flow requests, the greater the standard deviation. However, no matter which

Algorithm 2
Network partitioning based on K-means clustering.

```
Input: Network topology $G = (V, L)$
    Number of subnets $M$
    Maximum number of iterations $I_{\max}$
Output: Network partitioning results $G = \bigcap_{k=1}^{M} G_k = \bigcap_{k=1}^{M}(V_k, L_k)$
1    compute hops $h(v_i, v_j)$ between any two nodes, $\forall v_i, v_j \in V$
2    select $M$ nodes from set $V$ as centroids randomly
3.centroid set $cv = \{cv_1, cv_2, ..., cv_M\}$
4. for $I = 1$ to $I_{\max}$
5 initialization $V_k = \{\emptyset\}$, $k = 1, 2, 3, ..., M$
6 for $i = 1$ to $N$
7 compute distance between node $v_i$ and each centroid
8 $d_{ik} = h(v_i, cv_k), \forall v_i \in V, cv_k \in cv$
9 record node $v_i$ and centroid $cv_k$ corresponding to $\max d_{ik}$
10 $V_k = V_k \cup v_i$
11 $G_k = (V_k, L_k)$
12 endfor
13 for $k = 1$ to $M$
14 compute number $k^* = \left\lceil \sum_{v \in V_k} h(v, cv_k) / |G_k| \right\rceil$ and set $cv_{k^*}$ as new centroid
15 $cv_k = cv_{k^*}$
16 endfor
17 endfor
18 get the initial partition result $G_1, G_2, ..., G_M$
19 compute total flow requests of each subnet $G_k$: $TF_k = \sum_i \lambda_i v_i \in V_k$
20 if $TF_k > \max \beta(u) \cdot \omega(u)$
21 select the switch $v_j \in G_k$ with minimum flow request $(\min \lambda_j)$
22 select the neighbor subnet $G_l$ with $\min TF_l$
23 move $v_j$ from $G_k$ to $G_l$
24 update $G_k$ and $G_l$, and return Line 19
25.endif
26output $G = \bigcap_{k=1}^{M} G_k$
```

Algorithm 3
Heterogeneous controller placement based on improved GA.

```
Input:Subnets $G = \bigcap_{k=1}^{M} G_k = \bigcap_{k=1}^{M}(V_k, L_k)$
    Total types of heterogeneous controllers $u$
    Number of heterogeneous controllers $\{NC(1), ..., NC(u)\}$
    Evolution times $NE$
Output: Results of heterogeneous controller placement $c_k(u)$
1    rank all subnets in descending order based on the total flow requests of subnets
2    get the sets of sorted subnets $\{G_1, G_2, ..., G_M\}$ and set of all available controllers
3      for $k = 1$ to $M$
4        procedure genetic algorithm
5          for subnet $G_k$, initialize population
6          revise individuals of population
7          $fits \leftarrow F_k(population)$, $ne$=0
8          while ($ne$<$NE$)
9          $fit_{best} = \min(fits)$
10           $population_{new} \leftarrow$ Selection($population$)
11           $population_{new} \leftarrow$ Crossover($population_{new}$)
12           $population_{new} \leftarrow$ Mutation($population_{new}$)
13           revise individuals of $population_{new}$
14           $population \leftarrow population_{new}$
15           $fits \leftarrow F_k(population)$
16           if $fit_{best} \le \min(fits)$
17               $ne = ne + 1$
18             else $ne = 0$
19           endif
20         endwhile
21         decoding population
22         get the type and location of heterogeneous controller in $G_k$
23         output heterogeneous controller placement $c_k(u)$
24     endprocedure
25     update the number $NC(u) = NC(u) - 1$ and types of controllers available
26   endfor
```

kinds of topologies and how many flow requests, SQHCP always outperforms the other two approaches. The following reasons could account for these results. As the topology enlarges, the heterogeneous controllers require to manage more network nodes with different flow request rates. The complexity of the distributions of both nodes and flow requests aggravates the difficulty of balancing controller loads, causing the increase of standard deviation. Furthermore, as the number of flow requests increases, so does the probability that the heterogeneous controller overloads, exacerbating the standard deviation. Compared with other approaches, SQHCP computes the number of heterogeneous controllers based on topology and traffic characteristics, improves the heuristic algorithms to optimize the locations, and keeps reserved capacity for the controller to prevent overload. Therefore, SQHCP can always keep the standard deviation low and ensures the load balance of heterogeneous controllers.

5.2.5. *Performance of heuristic*

The design cores of SQHCP are improving heuristics algorithms. In order to show the performance of improved heuristics, we compare SQHCP with both optimum and original GA [32] to evaluate end-to-end delay optimization and running time, as shown in Fig. 24 and 25. Specifically, the optimum method lists all possible placement ways for heterogeneous controllers and searches optimal solution among them. The original GA method encodes the controller locations of the entire network and determines the types of heterogeneous controllers randomly.

In Fig. 24, the proposed SQHCP is close to Optimum in all three topologies, while Original GA has higher delays than SQHCP and Optimum. It indicates that our improved heuristics are effective and SQHCP could get the approximately optimal solution.

Fig. 25 shows the running time required to get the solution. Regardless of topologies, Optimum has the longest running time,

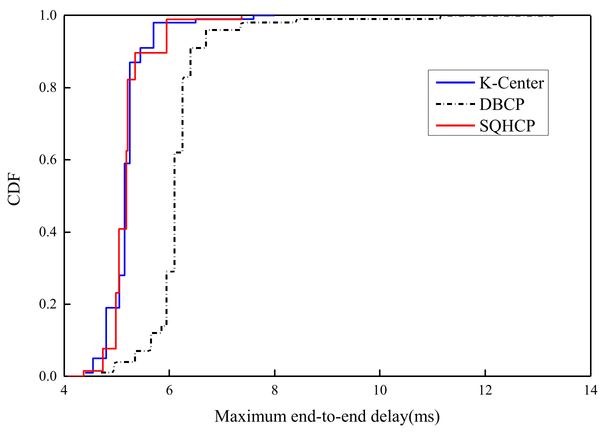

Fig. 17. CDF of maximum end-to-end delay in OS3E topology.

Original GA is second, and SQHCP is least. The reasons are explained as follows. Optimum adopts the method of exhaustion to find the optimal solution and its search space is extremely large, so its running time is longest. Besides, as the topology scale expands, the running time of Optimum increases exponentially. Compared with Original GA method, SQHCP divides the network into several subnets and implements GA for each subnet, which effectively shrinks the solution space and reduces the running time by 2.3 times at least. Moreover, the performance of SQHCP can be further improved by increasing iterations and optimizing GA parameters.

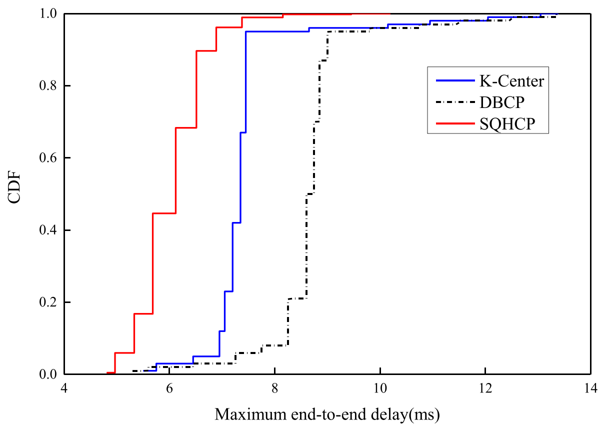

Fig. 18. CDF of maximum end-to-end delay in Columbus topology.

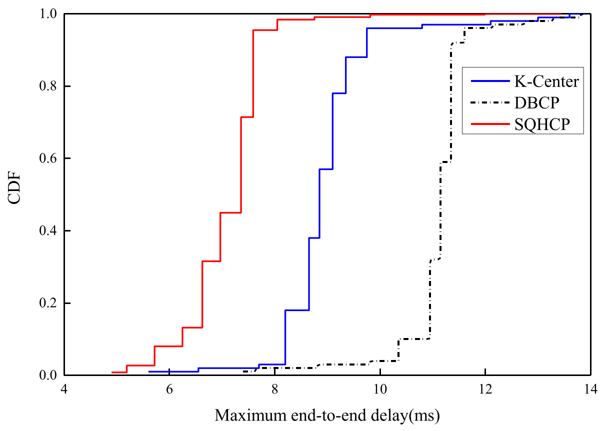

Fig. 19. CDF of maximum end-to-end delay in RF-II topology.

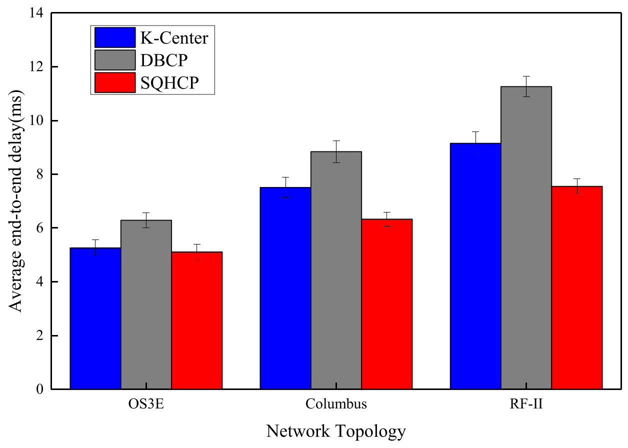

Fig. 20. The average end-to-end delay in different topologies.

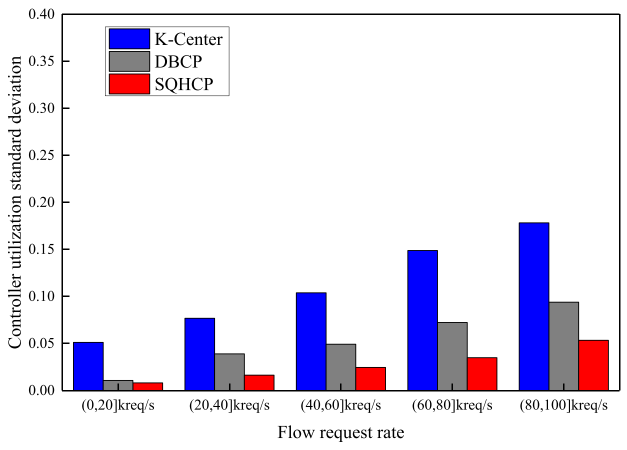

Fig. 21. Standard deviation of controller utilization in OS3E topology.

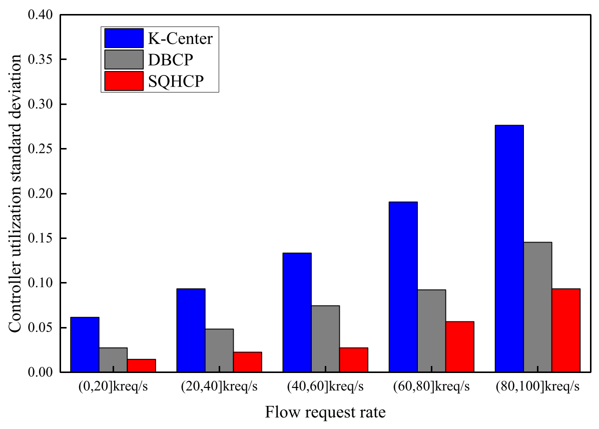

Fig. 22. Standard deviation of controller utilization in Columbus topology.

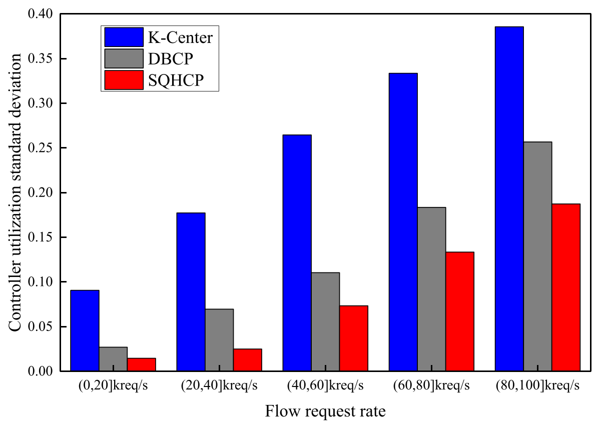

Fig. 23. Standard deviation of controller utilization in RF-II topology.

6. Related work

In recent years, researchers have proposed a variety of solutions to CPP in the SDN and achieved rich results. For example, in [54], the authors mainly discuss the controller placement approaches based on optimization objectives such as latency, connectivity, cost, load, energy, QoS, control plane overhead, or a combination of these objectives. In [55], the authors classify the CPP into several aspects (delay, reliability, cost, and multi-objective) and analyze specific algorithms in different application scenarios. Similar to the existing work, we will divide the CPP solutions into four categories: minimizing delay, minimizing deployment cost, minimizing fault rate, and multi-objective optimization. Each of these is described below.

Minimizing delay. Delay is crucial to SDN since switches and controllers interact with each other frequently. The existing CPP solutions usually focus on the propagation delay and controller processing delay.

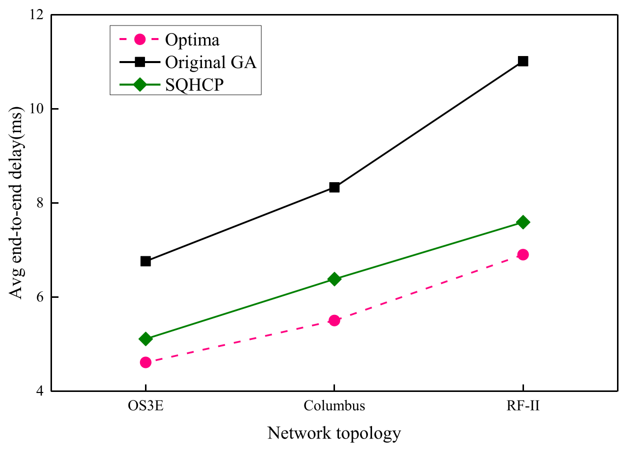

Fig. 24. Performance of heuristic in average end-to-end delay.

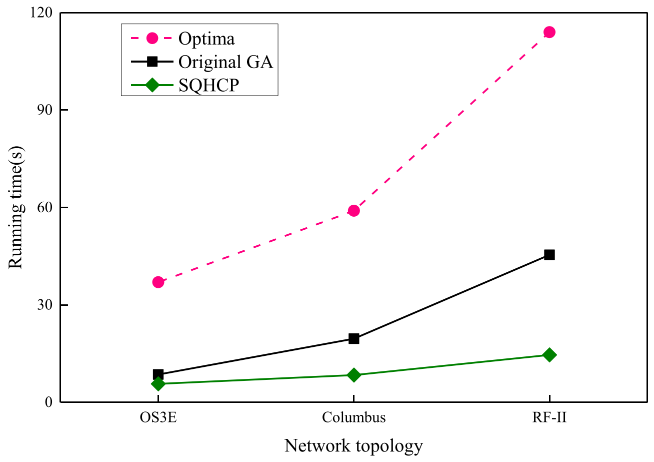

Fig. 25. Performance of heuristic in running time.

Considering the propagation delay, CPP is similar to the facility location problem. Heller et al. [9] first initiate the study on controller placement and give a clear definition for CPP: given a topology, how many controllers are needed, and where should they go? In the Internet 2 topology, they study the effects of controller placement on the average delay and worst-case propagation delay. Wang et al. [37] propose to transform the CPP into a network partition problem and use the clustering method to solve it. Zhu et al. [38] consider minimizing the propagation delay between switches and controllers and between controllers. They formulate the CPP as a control plane delay minimization problem and further propose a new algorithm based on clustering and Dijkstra to solve it. For the controllers with limited capacities, Yao et al. [10] define a capacitated CPP and design an efficient algorithm to optimize propagation delay.

Minimizing placement cost. With the large-scale application of SDN, how to reduce the costs of controller placement has attracted the attention of researchers. Afrim et al. [39] establish a mathematical model to solve this problem, with the goal of reducing network cost. Through the more detailed analysis, this model optimizes the number of controllers, locations and mapping relationships between network elements. Rath et al. [40] solve the CPP with the help of game theory. Based on the non-zero-sum game, the optimization engine of each controller calculates a revenue function, compares its revenue value with the neighbor's revenue value, and takes appropriate decisions (e.g., add or delete a controller). Alejandro et al. [41] introduce an energy-aware controller placement approach, which considers the maximum link utilization of each link when requiring to route the traffic. They formulate binary integer programming to reduce network energy consumption by closing links. Noelia Correia et al. [52] try to optimize the flow setup through reasonable controller placement, which considers both flow setup efficiency and possible future migrations from master to slave controllers. Further, the authors in [53] present a heuristic approach with solution shaking to solve the switch migration problem to balance controller loads. In this scheme, every switch migration is evaluated by how much benefit it will give to both the immigration and outmigration controllers.

Minimizing failure rate. Network node or link failure may lead to disconnections between switches and controllers, or even crash controller. Hence, how to determine the reasonable locations to place controllers to minimize the failure rate of the network is another hot potato. Hu et al. [42] study the reliability-optimized controller placement method and then prove it NP-hardness. Moreover, they define a new metric, the expected percentage of control path loss, to characterize the reliability of the control plane. Liao et al. [43] introduce a density cluster-based approach (DBCP) for the CPP. DBCP partitions the network into several subnets based on the density of switches and places that controller as closely as to other subnets. Zhong et al. [44] present a min-cover based controller placement approach to ensure the reliability of the SDN. They first give the definitions of neighborhood domain and minimum coverage set. The reliability and low latency of the control network are realized by placing as few controllers as possible. Hu et al. [45] propose a reliable and load balance-aware multi-controller deployment method, exploring a reliable controller deployment through balance node efficiency and path quality. Rao et al. [46] design an optimization model to improve the resilience of the controller placement. The objective is to minimize the backup capacity while keeping the total number of controllers be constant.

Multi-objective optimization. Recently, several researchers have considered many factors (e.g., delay, load balancing, and reliability) for solving CPP and formulate it as a multi-objective hybrid optimization problem. Stanislav et al. [11] consider delay, isolated node, and controller loads during the controller placement and propose a controller placement framework based on pareto optimization. They analyze all possible controller placement scenarios. Jalili et al. [12] consider the propagation delay, hop count, and link utilization in the CPP and further analyze the impact of these metrics on QoS. They present GA based on the analytic hierarchy process to compute the optimal controller placement. Wang et al. [47] study the CPP in the SDWAN and analyze the end-to-end delay and controller processing delay based on the queuing theory model.

Summary. Although there are plenty of research efforts on the CPP, almost all CPP solutions assume that the controllers are homogeneous and ignore the resulting common-mode fault. Therefore, we first propose to place heterogeneous controllers to solve the homogeneous controller common-mode fault and consider delay, controller utilization, and fault rate as the optimization objectives to enhance the security of the control plane and guarantee QoS.

7. Conclusion

In this paper, we introduce a new controller placement idea that deploys the heterogeneous controllers in the SDN to solve the common-mode fault problem of the existing CPP solutions. Firstly, we formally define HeCPP, considering end-to-end delay, controller utilization, and controller fault rate under the heterogeneous controllers. HeCPP is described as a multi-objective optimization problem, which aims to minimize network end-to-end delay, balance heterogeneous controller utilization, and lower control plane fault rate. In order to solve the HeCPP effectively, we propose SQHCP approach, which is secure-aware and QoS-guaranteed. SQHCP includes two steps. In step 1, we compute the types and the number of heterogeneous controllers required based on dynamic planning. In step 2, we first partition the network into several subnets based on the K-means clustering and then improve the genetic algorithm to optimize the heterogeneous controller placement for each subnet. On the one hand, we have proven the feasibility of SQHCP by several theorems. On the other hand, we have conducted extensive simulations to evaluate the performance of SQHCP under real topologies from the known Internet Topology Zoo. Simulation results verify that SQHCP can effectively enhance the security of the control plane and improve QoS compared with the existing typical approaches. In future work, we will make a further full investigation and then design a strategy that analyzes the heterogeneity and costs of controllers with different types to place controllers dynamically. Besides, we also plan to extend SQHCP approach to a practical network environment with more real traffic and trace real faults triggered by actual attacks and analyze the relationship between them in the practice.

Author statement

All authors (Peng Yi, Tao Hu, Yuxiang Hu, Julong Lan, Zhen Zhang, Ziyong Li) have seen and approved the final version of the manuscript being submitted. We warrant that the article is the authors' original work, hasn't received prior publication and isn't under consideration for publication elsewhere.

Declaration of Competing Interest

We declare that we have no financial and personal relationships with other people or organizations that can inappropriately influence our work, there is no professional or other personal interest of any nature or kind in any product, service and/or company that could be construed as influencing the position presented in, or the review of, the manuscript entitled.

Acknowledgment

This work is supported by the National Key Research and Development Project of China (Grant No. 2020YFB1804800), National Natural Science Foundation of China (Grant No. 62002382, 61802429, 61872382).

References

[1] D. Kreutz, F.M.V. Ramos, P.E. Veríssimo, C.E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-defined networking: a comprehensive survey, Proceed. IEEE 103 (1) (Jan. 2015) 14–76.
[2] J. Sushant, et al., B4: experience with a globally-deployed software defined WAN, in: Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM ACM, 2013.
[3] Guo Zehua, Xu Yang, et al., AggreFlow: Achieving Power Efficiency, Load Balancing, and Quality of Service in Data Center Networks, IEEE/ACM Transactions on Networking (2020), https://doi.org/10.1109/TNET.2020.3026015.
[4] S. Hassas Yeganeh, Y. Ganjali. "Kandoo: a Framework For Efficient and Scalable Offloading of Control Applications," in ACM HotSDN, 2012.
[5] T. Koponen, et al., Onix: a distributed control platform for large-scale production networks, in: 9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, BC, Canada, Proceedings USENIX Association, 2010.
[6] K. Phemius, M. Bouet, J. Leguay, DISCO: distributed SDN controllers in a multi-domain environment, in: 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, 2014, pp. 1–2.
[7] T. Hu, Z. Guo, P. Yi, T. Baker, J. Lan, Multi-controller based software-defined networking: a survey, IEEE Access 6 (2018) 15980–15996.
[8] G. Wang, Y. Zhao, J. Huang, W. Wang, The controller placement problem in software defined networking: a survey, IEEE Netw. 31 (5) (2017) 21–27.
[9] Brandon Heller, R. Sherwood, N. McKeown, The controller placement problem. Workshop On Hot Topics in Software Defined Networks ACM, 2012.
[10] G. Yao, J. Bi, Y. Li, L. Guo, On the capacitated controller placement problem in software defined networks, IEEE Commun. Lett. 18 (Aug. (8)) (2014) 1339–1342.
[11] S. Lange, et al., Heuristic approaches to the controller placement problem in large scale SDN networks, IEEE Trans. Netw. Serv. Manage. 12 (March (1)) (2015) 4–17.
[12] J. Ahmad, et al., Multi criteria analysis of controller placement problem in software defined networks. Computer Communications, 2019.
[13] K, Benzekki, A, E, Fergougui, A,E, Elalaoui, Software-defined networking (SDN): a survey. Security & Communication Networks, 2017.
[14] R.P.K. Christian, T. Holz, On network operating system security, Int. J. Netw. Manage. (2016).
[15] D. Tatang, F. Quinkert, J. Frank, C. Röpke, T. Holz, SDN-guard: protecting SDN controllers against SDN rootkits, in: 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Berlin, 2017, pp. 297–302.
[16] Si Xueming, et al., A review of the basic theory of mimic defense. Strategic Study of CAE, 2016.
[17] A. Dixit, K. Kogan, P. Eugster, Composing heterogeneous SDN Controllers with flowbricks, in: IEEE International Conference on Network Protocols IEEE, 2014.
[18] Qi Chao, et al., An intensive security architecture with multi-controller for SDN, IEEE Infocom Workshops (2016).
[19] Hongchao Hu, et al., MNOS: a mimic network operating system for software defined networks, IET Inf. Sec. 11 (6) (2017) 345–355.
[20] H. Maziku, S. Shetty, D. Jin, C. Kamhoua, L. Njilla, K. Kwiat, Diversity modeling to evaluate security of multiple SDN controllers, in: 2018 International Conference on Computing, Networking and Communications (ICNC), Maui, HI, 2018, pp. 344–348.
[21] J. Voas, A. Ghosh, F. Charron, L. Kassab, Reducing uncertainty about common-mode failures, in: Proceedings the Eighth International Symposium on Software Reliability Engineering, Albuquerque, NM, USA, 1997, pp. 308–319.
[22] Avizienis, Kelly, Fault tolerance by design diversity: concepts and experiments, Comput. (Long Beach Calif) 17 (Aug. (8)) (1984) 67–80.
[23] Y.E. Oktian, et al., Distributed SDN controller system: a survey on design choice, Comput. Netw. (2017) 100–111.
[24] L. Mamushiane, A. Lysko, S. Dlamini, A comparative evaluation of the performance of popular SDN controllers. Wireless Days (WD), Dubai, 2018, pp. 54–59.
[25] R. Khondoker, A. Zaalouk, R. Marx, K. Bayarou, Feature-based comparison and selection of software defined networking (SDN) controllers, in: 2014 World Congress on Computer Applications and Information Systems (WCCAIS), Hammamet, 2014, pp. 1–7.
[26] Z. Guo, W. Chen, Y. Liu, Y. Xu, Z. Zhang, Joint switch upgrade and controller deployment in hybrid software-defined networks, IEEE J. Sel. Areas Commun. 37 (May (5)) (2019) 1012–1028.
[27] T. Hu, P. Yi, J. Zhang, J. Lan, A distributed decision mechanism for controller load balancing based on switch migration in SDN, China Commun. 15 (Oct. (10)) (2018) 129–142.
[28] K. Bu, X. Wen, B. Yang, Y. Chen, L.E. Li, X. Chen, Is every flow on the right track?: inspect SDN forwarding with RuleScope, in: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, 2016, pp. 1–9.
[29] A.A. Kasi, F. Khan, B.A. Ahmed, S. Rashid, S. Waseem, Performance analysis of homogenous and heterogeneous network core switches, in: International Symposium on Wireless Systems and Networks (ISWSN), Lahore, 2017, pp. 1–7.
[30] G. Borradaile, B. Heeringa, G. Wilfong, The knapsack problem with neighbour constraints. J. Discrete Algorithms, 2012, pp. 224–235.
[31] T. Henri. "A Note on Certainty Equivalence in Dynamic Planning," HenriTheil's Contributions to Economics and Econometrics. 1992.
[32] B. Babayiğit, B. Ulu, E.N. Hasçokadar, Solving multi-controller placement problem in software defined networks with a genetic algorithm, in: 2019 4th International Conference on Computer Science and Engineering (UBMK), Samsun, Turkey, 2019, pp. 666–670.
[33] V. Huang, G. Chen, Q. Fu, E. Wen, Optimizing controller placement for software-defined networks, in: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Arlington, VA, USA, 2019, pp. 224–232.
[34] H. Xiaolan, W. Muqing, X. Weiyao, A controller placement algorithm based on density clustering in SDN, in: 2018 IEEE/CIC International Conference on Communications in China (ICCC), Beijing, China, 2018, pp. 184–189.
[35] S. Knight, H.X. Nguyen, N. Falkner, R. Bowden, M. Roughan, The internet topology zoo, IEEE J. Sel. Areas .Commun. 29 (October (9)) (2011) 1765–1775.
[36] A. Ruiz-Rivera, K. Chin, S. Soh, GreCo: an energy aware controller association algorithm for software defined networks, IEEE Commun. Lett. 19 (April (4)) (2015) 541–544.
[37] G. Wang, Y. Zhao, J. Huang, Q. Duan, J. Li, A K-means-based network partition algorithm for controller placement in software defined network, in: 2016 IEEE International Conference on Communications, Kuala Lumpur, 2016, pp. 1–6.
[38] L. Zhu, R. Chai, Q. Chen, Control plane delay minimization based SDN controller placement scheme, in: 2017 9th International Conference on Wireless Communications and Signal Processing (WCSP), Nanjing, 2017, pp. 1–6.
[39] A. Sallahi, M. St-Hilaire, Optimal model for the controller placement problem in software defined networks, IEEE Commun. Lett. 19 (Jan. (1)) (2015) 30–33.
[40] H.K. Rath, V. Revoori, S.M. Nadaf, A. Simha, Optimal controller placement in software defined networks (SDN) using a non-zero-sum game, in: Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014, Sydney, NSW, 2014, pp. 1–6.
[41] A. Ruiz-Rivera, K. Chin, S. Soh, GreCo: an energy aware controller association algorithm for software defined networks, IEEE Commun. Lett. 19 (April (4)) (2015) 541–544.
[42] Y. Hu, W. Wang, X. Gong, X. Que, S. Cheng, On reliability-optimized controller placement for software-defined networks, China Commun. 11 (2) (Feb 2014) 38–54.
[43] Jianxin Liao, et al., Density cluster based approach for controller placement problem in large-scale software defined networkings, Comput. Netw. (2017).
[44] Q. Zhong, Y. Wang, W. Li and X. Qiu, "A min-cover based controller placement approach to build reliable control network in SDN," in NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium, Istanbul, 2016, pp. 481–487.

[45] T. Hu, P. Yi, J. Zhang, J. Lan, Reliable and load balance-aware multi-controller deployment in SDN, China Commun. 15 (Nov. (11)) (2018) 184–198.
[46] K.B.P. Rao, S.V. Rao, Towards improving resilience of controller placement with minimum backup capacity in software defined networks. Comput.Netw., 2019.
[47] G. Wang, Y. Zhao, J. Huang, Y. Wu, An effective approach to controller placement in software defined wide area networks, IEEE Trans. Netw. Serv. Manage. 15 (March (1)) (2018) 344–355.
[48] Jinbao Yang, et al., Invulnerability analysis for SDN based on complex network theory, in: International Conference on Human Centered Computing, Springer International Publishing, 2016.
[49] P. Kampanakis, H. Perros, T. Beyene, SDN-based solutions for moving target defense network protection, in: 2014 IEEE 15th WoWMoM IEEE, 2014.
[50] T. Wang, F. Liu, H. Xu, An efficient online algorithm for dynamic SDN controller assignment in data center networks, IEEE/ACM Trans. Netw. 25 (Oct. (5)) (2017) 2788–2801.
[51] T. Amin, Y. Ganjali, Hyper flow: a distributed control plane for open flow, in: Internet Network Management Conference on Research on Enterprise Networking USENIX Association, 2011.
[52] F. Al-Tam, N. Correia, On load balancing via switch migration in software-defined networking, IEEE Access 7 (2019) 95998–96010.
[53] K.B.P. Rao, S.V. Rao, Controller placement in software defined networks: a comprehensive survey, Comput. Netw. (163) (2019).
[54] J. Lu, Z. Zhang, T. Hu, P. Yi, J. Lan, A survey of controller placement problem in software-defined networking, IEEE Access 7 (2019) 24290–24307.
[55] Z. Bang, X. Wang, M. Huang, Multi-objective optimization controller placement problem in internet-oriented software defined network, Comput. Commun. (2018).
[56] W.U. Qi, C. Hongchang, C. Fucai, Security analysis in heterogeneous fault-tolerant control plane, Chin. J. Netw. Inf. Sec. (2018).
[57] C. Qi, J. Wu, H. Chen, H. Yu, H. Hu and G. Cheng, "Game-theoretic analysis for security of various software-defined networking (SDN) architectures," 2017 IEEE 85th Vehicular Technology Conference (VTC Spring), Sydney, NSW, 2017, pp. 1–5.
[58] T.E. Davis, J.C. Principe, A markov chain framework for the simple genetic algorithm, Evol Comput 1 (Sept. (3)) (1993) 269–288.
[59] Common vulnerabilities and exposures, http://cve.mitre.org/, 2020.
[60] National vulnerability database, https://nvd.nist.gov/, 2020.
[61] G.N. Nde, R. Khondoker, SDN testing and debugging tools: a survey, in: 2016 5th International Conference on Informatics, Electronics and Vision (ICIEV), Dhaka, 2016, pp. 631–635.
[62] M.P. Novaes, L.F. Carvalho, J. Lloret, M.L. Proença, Long short-term memory and fuzzy logic for anomaly detection and mitigation in software-defined network environment, IEEE Access 8 (2020) 83765–83781.
[63] Qi Chao, et al., Security Analysis of Dynamic SDN Architectures Based On Game Theory, Security and Communication Networks, 2018, pp. 1–10.
[64] C. Coello, A. Coello, Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art, Comput. Methods Appl. Mech. Eng. 191 (2002) 1245–1287.

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.

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, data plane and network security.

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.

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.

Zhen Zhang is an assistant professor of National Digital Switching System Engineering and Technological Research Center. His-contributions encompass aspects of network architecture and network security.

Ziyong Li was born in 1995. He is currently pursuing the Doctor degree at Information Engineering University, Zhengzhou, China. He received the B.E. degree in Harbin Institute of Technology in 2017. His-research interests include new generation information communication network, Software-defined networking and network security.