Computer Communications 163 (2020) 35–50

Contents lists available at ScienceDirect

Computer Communications

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

Controller placements for latency minimization of both primary and backup paths in SDNs

Yuqi Fan [a,*], Lunfei Wang [a], Xiaohui Yuan [b]

[a] School of Computer Science and Information Engineering, Anhui Province Key Laboratory of Industry Safety and Emergency Technology, Hefei University of Technology, Hefei, Anhui, 230601, China

[b] Department of Computer Science and Engineering, University of North Texas, Denton, TX, 76203, USA

ARTICLE INFO

Keywords:
SDN
Controller placement
Single link failure
Reliability
Multi-objective optimization
Latency

ABSTRACT

Software-defined networking (SDN) is a revolutionary network architecture that separates the network control layer from the underlying equipment. Multiple controllers form a logically centralized control layer in large-scale networks, which raises the controller placement problem. Most of the research on latency-oriented controller placement optimized the delay between switches and controllers assuming the network is reliable. However, the network is subject to link failures. In this paper, we formulate a novel multi-objective SDN controller placement problem with the aim to minimize the switch-to-controller communication delay for both the cases without link failure and with single-link-failure. We propose an efficient metaheuristic-based Reliability-Aware and Latency-Oriented controller placement algorithm (RALO) for multi-objective multiple controller placements. The algorithm constructs an initial feasible solution by a greedy method with network partition, then repeatedly generates new solutions with variable neighborhood search. Once a new solution is generated, the algorithm decides whether to accept the new solution as a non-dominated solution to the problem and performs update operation on the Pareto optimal solution set. Meanwhile, to avoid falling into the local optimum, the algorithm also performs perturbation and destruction operations on the current solution. We finally conduct experiments through simulations on 8 real network topologies and two kinds of generated networks conforming to ER (Erdos–Renyi) random model and small-world model. Experimental results demonstrate that the proposed algorithm can achieve a competitive performance of switch-to-controller latencies in both the cases without link failure and with single-link failure, and the accumulated delay of primary and backup paths between the controllers and the switches. The Pareto optimal solution set provided by algorithm RALO allows network administrators with flexible choices to strike a trade-off between the switch-to-controller delay of primary and backup paths.

1. Introduction

Software-defined networking (SDN) is a revolutionary network architecture that separates the network control layer from underlying equipment [1]. In SDNs, network switches (nodes) are only responsible for data forwarding, while controllers determine the path of network packets across the switches. Upon the arrival of an unknown flow, the switch sends a flow set-up request to the controller which responds to the request with a flow entry to be installed in the flow table of the switch.

The software platform located in the centralized controller implements programmable control of the underlying hardware and achieves flexible on-demand distribution of network resources [2]. However, the constraints of processors, memory, access bandwidth, and other resources make it infeasible to implement the logical control layer with only one controller in large-scale networks. Furthermore, the control layer with one controller also faces the single-point-failure problem. To address these issues, multiple-controller architectures, such as Kandoo [3], HyperFlow [4], Onix [5], have been introduced. The multiple-controller architecture also raises a new problem, the controller placement problem, which decides the controllers' positions and the mapping relationship between the controllers and the nodes, since random placement is far from optimal [6,7]. The controller placement decisions have an important impact on the network performance in terms of delay, reliability, etc.

In addition to the switch-to-controller latency which is critical for the performance of SDNs, reliability is an important concern for SDNs, since network failures can cause disconnections between the switches and the controllers [8], and even incur cascading failures of other controllers [9]. Upon the link failure, a backup path needs to be set up

* Corresponding author.
E-mail addresses: yuqi.fan@hfut.edu.cn (Y. Fan), cxwlf@mail.hfut.edu.cn (L. Wang), xiaohui.yuan@unt.edu (X. Yuan).

https://doi.org/10.1016/j.comcom.2020.09.001
Received 20 February 2020; Received in revised form 2 July 2020; Accepted 1 September 2020
Available online 11 September 2020

between the controllers and the switches. Note that the two objectives of primary path delay and backup path latency are intrinsically conflicting with each other [10]. The controllers are prone to be placed close to the switches to reduce the switch-to-controller latency on the primary paths assuming the network is ideal. However, the backup path delay may be high due to long detour paths when a link failure occurs.

Most of the research on latency-oriented controller placement optimizes the delay between the switches and the controllers with the assumption of a reliable network available. In this paper, we tackle the controller placement problem to minimize the latency of both primary and backup paths between the controllers and the switches, assuming that there is at most one link failure in the network [11].

The main contributions of this paper are as follows. We address the controller placement problem to reduce the latency of the flow set-up requests with and without single-link-failure. We then formulate a novel multi-objective SDN controller placement problem to minimize the switch-to-controller communication delay for both the cases without link failure and with single-link-failure simultaneously. We propose an efficient metaheuristic-based Reliability-Aware and Latency-Oriented controller placement algorithm (RALO) for multi-objective multiple controller placement. The algorithm constructs an initial feasible solution by a greedy method with network partition, then repeatedly generates new solutions with variable neighborhood search. Once a new solution is generated, the algorithm decides whether to accept the new solution as a non-dominated solution to the problem and performs update operation on the Pareto optimal solution set. Meanwhile, to avoid falling into the local optimum, the algorithm performs perturbation and destruction operations on the current solution. The algorithm obtains a set of non-dominated solutions as the approximate solutions of the precise Pareto frontier. We finally conduct experiments through simulations on ATT and Internet2 networks and two kinds of generated networks conforming to ER (Erdos–Renyi) random model and small-world model. Experimental results demonstrate that the proposed algorithm can achieve competitive performance in terms of the switch-to-controller latencies in both the cases without link failure and with single-link failure, and the accumulated switch-to-controller delay of the primary and the backup paths between the controllers and the switches.

The rest of the paper is organized as follows. Section 2 introduces the related work. The problem is formulated in Section 3. Section 4 presents the proposed algorithm. The performance evaluation is given in Section 5, and Section 6 concludes the paper.

2. Related work

The switch-to-controller latency is critical for the performance of SDNs, and some research has been conducted on the controller placement problem with the objective of minimizing the switch-to-controller latency. Heller et al. first proposed the controller placement problem to minimize the communication latency between the switches and the controllers by deciding the number and locations of controllers [6]. Bari et al. designed a framework to tackle the problem of multiple controller placements within a WAN; according to the network state, the framework dynamically adjusts the number of active controllers and delegates a subset of switches to each controller [12]. Rath et al. introduced an approach to reducing the communication latency and increase the utilization of the controllers based on game theory; the approach decides when to add or shut down controllers dynamically [13]. Tanha et al. formulated the resilient capacitated controller placement problem which takes both the switch-controller/inter-controller latency requirements and the capacity of the controllers into account and proposed two algorithms for the problem [14]. Yao et al. introduced a latency metric to minimize the total cost of flow-setup request from the switches to the controllers to deal with the mapping between the switches and the controllers under dynamic flow variations; the metric considers the weight of switches and the delay from the switches to the controllers simultaneously, where the weight of a switch is related to the node degree of the switch and the maximum node degree in the network [15]. Sallahi et al. proposed a model to simultaneously determine the optimal number, locations, and type of controller(s) as well as the interconnections between all the network nodes by considering the capacity of the controllers and path delay [16]. Huque et al. proposed a solution to combine a controller placement algorithm with a dynamic flow management algorithm, where the locations of controller modules are determined by the switch-to-controller latency constraint and the number of the controllers at each controller module is adapted according to the traffic load [17]. Wang et al. designed a scheme to minimize the maximum latency between the controllers and the associated switches based on network partition; the scheme divides the network into multiple subnetworks with an improved k-means algorithm and a controller is deployed in each subnetwork to minimize the maximum latency between the controller and the associated switches in the subnetwork [18]. Jia et al. proposed an algorithm to provide policy-aware unicast request admissions with and without end-to-end delay constraints in a software-defined network to minimize the operational cost of admitting a single request in terms of both computing resource consumption and bandwidth resource consumption for routing its data traffic, or maximizing the network throughput for a sequence of requests without the knowledge of future request arrivals [19].

Reliability is also a key problem in controller placements. Hu et al. introduced a metric called expected percentage of control path loss due to a failed network component to characterize the reliability of SDN networks and proposed a heuristic algorithm l-w-greedy to analyze the trade-off between reliability and latency; the expected percentage of control path loss is related to the number of control paths going through a component and the failure probability of the component [20,21]. Guo et al. studied the impact of controller placement on network resilience with interdependence graphs and cascading failure analysis and proposed a partition and selection approach to controller placements to improve the resilience [22]. Hock et al. introduced a resilience framework to cope with the resilience of link outages and proposed a Pareto-based optimal controller placement method (POCO) to maximize node-to-controller latencies and resilience in terms of failure tolerance and load balancing [23]. Lange et al. extended the POCO framework with heuristics to support large-scale networks or dynamic networks with properties changing over time [24]. Müller proposed a controller placement strategy, Survivor, to explore path diversity to optimize the survivability of networks to maximize the number of node-disjoint paths between the switches and the controllers; the strategy enhances connectivity by explicitly considering path diversity [25]. Dorabella et al. formulated a robust controller placement problem variant to maximize the network robustness for a given number of malicious node attacks and proposed an ILP based method to enumerate all solutions [26]. Vizarreta et al. proposed two strategies to improve the reliability by using redundant communication paths and backup controllers; with the first strategy, the switches are connected to a controller over two disjoint control paths; the second strategy makes switches be connected to two different controller replicas over two disjoint paths [27]. Ros et al. guaranteed the reliability of controller placements by making each switch be connected to multiple controllers [28]. Huang et al. studied a weighted cost-minimization problem to reduce the congestion in the neighboring links of the failed link and control-channel setup cost [29]. Fan et al. proposed a latency-aware reliable controller placement algorithm LARC by jointly taking into account both the communication reliability and the communication latency between the controllers and the switches if any link in the network fails [30].

Previous research has shown that both delay and reliability are important for SDNs. When a link fails in the network, we need to establish a backup path for the transmission of flow set-up requests and responses between the controllers and the switches [27]. We use

the shortest path which bypasses the failed link [27] to connect the switch and its associated controller. Note that the two objectives of the primary path delay and the backup path latency are intrinsically conflicting with each other [10]. However, most existing research on latency-oriented controller placement optimizes the delay between the switches and the controllers with the assumption of a reliable network available. In this paper, we study the problem of minimizing the latency in both primary and backup paths between the controllers and the switches, assuming the single-link failure may happen.

3. Problem formulation

An SDN network topology can be represented as a graph $G = (V, E)$, where V is the finite set of switches and E is the set of links between the nodes. $K(K \leq |V|)$ controllers are to be placed in the network. Each controller is co-located with one and only one switch [23], and the total number of requests processed by each controller should be within its processing capacity. Each switch is mapped to exactly one controller; that is, the requests of a switch cannot be split to be processed by multiple controllers. When a switch is mapped to a controller, we say the switch and the controller are associated with each other. We assume that there is at most one link failure in the network because the possibility of multi-link failures in the network is low [11]. The notations used in the paper are listed in Table 1.

The primary path $p_{i,k}$ between switch i and controller c_k is the shortest path between these two nodes. The average latency in all the primary paths in the network can be calculated via Eq. (1).

$$l^p = \frac{\sum_{i \in V} \sum_{c_k \in C} l^p_{i,k} * x_{i,k}}{N}. \tag{1}$$

Upon the failure of link (i', j') on primary path $p_{i,k}$, we need to set up a backup path to connect switch i and controller c_k. We use the shortest path which bypasses the failed link (I', j') to rebuild the connection between switch i and its associated controller c_k. Since every link on primary path $p_{(i,k)}$ may fail, we calculate the average backup path latency between switch i and controller c_k via Eq. (2). Considering all single link failures in the network, the average latency of all the backup paths in the network is calculated by Eq. (3).

$$l^b_{i,k} = \frac{\sum_{(i',j') \in p_{i,k}} l^b_{i,k,i',j'}}{|p_{i,k}|}. \tag{2}$$

$$l^b = \frac{\sum_{i \in V} \sum_{c_k \in C} l^b_{i,k} * x_{i,k}}{N}. \tag{3}$$

In this paper, we aim to determine where to place each controller and the exact association relationship between the controllers and the switches, to optimize both the average primary path latency and the average backup path latency. In other words, our optimization objective is to minimize

$$[l^p, l^b], \tag{4}$$

Subject to

$$\sum_{k=1}^{K} x_{i,k} = 1, \quad \forall i \in V, \tag{5}$$

$$\sum_{i=1}^{N} y_{i,k} = 1, \tag{6}$$

$$y_{i,k} \leq x_{i,k}, \quad \forall i \in V, \tag{7}$$

$$\sum_{i=1}^{N} x_{i,k} r_i \leq u_k, \tag{8}$$

$$x_{i,k} \in \{0, 1\}, \quad y_{i,k} \in \{0, 1\}. \tag{9}$$

Eq. (5) ensures that each switch is assigned to exactly one controller. Eq. (6) mandates that each controller is co-located with one and only one switch. Eq. (7) dictates that switch i is mapped to controller c_k if controller c_k is co-located with switch i. Eq. (8) signifies that the number of requests assigned to the controller cannot exceed the processing capacity of the controller. Eq. (9) ensures that $x_{i,k}$ and $y_{i,k}$ are binary integer variables.

4. Controller placement algorithm

It is known that the controller placement problem to minimize the latency of primary paths is an $\mathcal{NP}$-complete problem [6]. The controller placement problem in this paper is $\mathcal{NP}$-complete too, as the former is a special case of our problem by ignoring the optimization of the latency of backup paths.

In this section, we propose an efficient metaheuristic-based Reliability-Aware and Latency-Oriented controller placement algorithm (RALO) for the multi-objective controller placement problem. The algorithm constructs an initial feasible solution by dividing the network nodes into multiple switch subsets and allocating a controller in each switch subset according to the switch-to-controller communication delay and controller processing capacity since it is shown that network partitioning can help controller placements [18]. The algorithm then repeatedly generates new solutions with variable neighborhood search (VNS). Meanwhile, to avoid falling into the local optimum, the algorithm performs perturbation and destruction operations on the current solution.

For a feasible solution X', we decide whether to accept solution X' as a non-dominated solution to the problem and performs *Update* operation on the Pareto optimal solution set $\mathcal{X}$ according to the following rules:

1. If solution X' performs better in both of the goals than any existing Pareto optimal solution X'', Algorithm 1 adds solution X' into set $\mathcal{X}$ as a Pareto optimal solution, and deletes solution X'' from Pareto optimal solution set $\mathcal{X}$.
2. If solution X' only performs better in one of the two objectives than all the existing Pareto optimal solutions, Algorithm 1 adds solution X' as a Pareto optimal solution into set $\mathcal{X}$.
3. If solution X' performs worse in both of the two goals than any existing Pareto optimal solutions, solution X' is dropped.

The controller placement algorithm RALO shown in Algorithm 1 consists of five components: *initial solution construction (division and construction), VNS, perturbation, controller relocation*, and *shake*. In each component, once a new solution is generated, the algorithm conducts update operation on the Pareto optimal solution set.

Algorithm 1 starts with the construction of an initial feasible solution. The construction is divided into two steps: node subset division (step 2) and controller placement in each node subset (step 3). Algorithm 1 then generates new solutions and accepts the new feasible non-dominated solutions to the problem by the operations of perturbing, destroying the current solution, changing controller location, local search, etc.

Algorithm 1 proceeds to find new solutions iteratively within a maximum number of iterations determined by the parameter r_{max}. During each iteration, the mapping relationship between the controllers and the switches is perturbed (step 6) and the variable neighborhood search is performed (step 7), followed by the positions change of the controllers (step 8). If the current solution is worse than all of the found Pareto optimal solutions, the current solution is perturbed greatly by *shake* operation so that the solution can jump to another solution space to avoid falling into the local optimum (steps 9–11).

4.1. Initial solution construction

The initial feasible solution is carefully constructed with two steps: node subset division and controller placement in each subset, as illustrated in Algorithm 2 and Algorithm 3, respectively.

Algorithm 2 divides the network nodes into K subsets by finding K potential controller locations in turn. Assume C' is the set of potential controller locations. Initially, a random node i is chosen as the first potential controller location, and $C' = \{i\}$ (step 1). For each node,

Table 1
Table of notations.

Notation	Definition
N	The number of nodes/switches ($N = \|V\|$)
C	Controller set
K	The number of controllers ($K \le N$)
i, j	Switches/nodes i and j
(i, j)	The link between nodes i and j
r_i	The number of requests from switch i ($1 \le i \le N$)
c_k	Controller c_k
u_k	The processing capacity of controller c_k ($1 \le k \le K$)
$x_{i,k}$	Indicate whether switch i is mapped to controller c_k (= 1) or not (= 0)
$y_{i,k}$	Denote whether controller c_k is co-located with switch i (= 1) or not (= 0)
$p_{i,k}$	The primary path between switch i and controller c_k
$l^p_{i,k}$	The latency of the primary path between switch i and controller c_k
$l^b_{i,k,i',j'}$	The latency of the backup path between switch i and controller c_k under link (i', j') failure
$l^b_{i,k}$	The average latency of the backup paths between switch i and controller c_k
X'	A feasible solution
X'_k	The subset of switches mapped to controller c_k in feasible solution X'
$\ell(X')$	The set of switches whose delay to the associated controller is the largest in its corresponding switch subset
H^i	The set of switches which can be reached by switch i in two hops
$\mathcal{X}$	Pareto optimal solution set

Algorithm 1 *Controller Placement Algorithm RALO*

Input: Network topology $G = (V, E)$, the number of requests of the switches, the number of controllers K, the number of iterations r_{max}, parameters α, β.
Output: Pareto optimal solution set $\mathcal{X}$.

$\mathcal{X} = \emptyset$;
S = Division(G, K);
X' = Construction$(G, S, K, \mathcal{X})$;
$r = r_{max}$;
while $r > 0$ **do**
  Perturbation (X', α);
  X' = VNS$(X', \beta, \mathcal{X})$;
  X' = Relocation$(X', \mathcal{X})$;
  if X' is worse than all the solutions in $\mathcal{X}$ **then**
    X' = Shake(X', η);
  end if
  $r = r - 1$;
end while
return $\mathcal{X}$.

we find L_i, i.e., the delay of the node to the closet potential controller position as follows:

$$L_i = \min_{c_k \in C'} l^p_{i,k}, \quad \forall i \in V. \tag{10}$$

Every time after a potential controller location is found, Algorithm 2 updates L_i for all the nodes, and then chooses the node with maximum L_i as the next potential controller location; that is, the chosen node $i^* = \arg\max_{i \in V - C'} L_i$. Algorithm 3 repeats the process of selecting the node with maximum L_i as a new potential controller location (step 6) until we get all the K potential locations. Each of the K nodes forms a subset S_k; that is, subset S_k contains only one node k. Algorithm 3 assigns each node i to subset S_k which causes the minimum cost from node i to the kth potential controller location (steps 12–15).

Algorithm 3 chooses the location for each controller in each node subset obtained by Algorithm 2, and maps each node to a controller under the controller processing capacity constraint. It is advantageous to deploy the controller on a node with a large number of requests from nearby nodes. In Algorithm 3, the node with the maximum number of requests from the nodes in two hops is chosen as the controller location in each subset (steps 4–7). The algorithm sorts all the other nodes in each subset S_k by the non-ascending order of the number of requests (step 9) and assigns each node to the controller under the controller capacity constraint (steps 10–14). For the nodes that cannot be assigned

Algorithm 2 *Division*

Input: Network topology $G = (V, E)$, the number of controllers K.
Output: Subset division $S = \{S_1, S_2, \cdots, S_k, \cdots, S_K\}$.

1: Choose $i \in V$; $C' = \{i\}$; $k = 1$; $S_k = \{i\}$;
2: **for** each $i \in V - C'$ **do**
3: $L_i = l^p_{i,k}, c_k \in C'$;
4: **end for**
5: **while** $|C'| < K$ **do**
6: $i^* = \arg\max_{i \in V - C'} L_i$;
7: $k = k + 1$; $S_k = \{i^*\}$; $C' = C' \cup \{i^*\}$;
8: **for** each $i \in V - C'$ **do**
9: $L_i = \min_{c_k \in C'} l^p_{i,k}$;
10: **end for**
11: **end while**
12: **for** each $i \in V - C'$ **do**
13: $k = \arg\min_{k' \in C'} l^p_{i,k'}$;
14: $S_k = S_k \cup \{i\}$;
15: **end for**
16: **return** S.

to a controller, the algorithm maps each of the unassigned nodes to the closest (least primary paths delay) controller which has enough processing capacity to serve the requests from the node (steps 16–18). After finding the new solution, the algorithm performs *Update* operation on the Pareto optimal solution set (step 19).

4.2. Variable neighborhood search

VNS shown in Algorithm 4 searches the neighborhood of the current solution to generate new feasible solutions. We denote $\mathcal{L}_1(X', \beta)$ and $\mathcal{L}_2(X', \beta)$ as the set consisting of the nodes whose latencies of the primary paths and the backup paths to the associated controller are among the top β largest in the corresponding node subset, respectively. For nodes in $\mathcal{L}_s(X', \beta)$ $(s \in \{1, 2\})$, we define two operations: remap(i, k, q) and swap(i, j).

Operation remap(i, k, q) reassigns switch $i \in \mathcal{L}_s(X', \beta)$ $(s \in \{1, 2\})$ from the originally assigned controller c_k to another controller c_q $(k \neq q)$ to generate a new solution, and the benefit of the remapping operation is defined by Eqs. (11) and (12).

$$\pi_1^1(i, k, q) = l_{i,k}^p - l_{i,q}^p. \tag{11}$$

$$\pi_1^2(i, k, q) = l_{i,k}^b - l_{i,q}^b. \tag{12}$$

where $\pi_1^1(i, k, q) > 0$ and $\pi_1^2(i, k, q) > 0$ indicate that the delay on the primary paths and the backup paths will be reduced, respectively, if switch i originally assigned to controller c_k is remapped to controller c_q.

Operation swap(i, j) remaps switch i originally assigned to controller c_k to controller c_q, and reassigns switch j $(i \neq j)$ originally mapped to controller c_q to controller c_k $(k \neq q)$. The benefit of the swap operation is defined by Eqs. (13) and (14). If $\pi_2^s(i, j, k, q) > 0$ $(s \in \{1, 2\})$, we can decrease the delay by swapping the mapping relationship between the switches and the controllers.

$$\pi_2^1(i, j, k, q) = (l_{i,k}^p + l_{j,q}^p) - (l_{i,q}^p + l_{j,k}^p). \tag{13}$$

$$\pi_2^2(i, j, k, q) = (l_{i,k}^b + l_{j,q}^b) - (l_{i,q}^b + l_{j,k}^b). \tag{14}$$

Note that the two neighborhood search operations will be executed only when $\pi_r^s(i, j) > 0$ $(s, r \in \{1, 2\})$. Once the neighborhood search operations are executed, a new solution is generated, and the algorithm performs update operation on the Pareto solution set $\mathcal{X}$ (steps 11 and 19). In step 24, the solution with less average primary path delay is considered to be a better solution when $s = 1$. Similarly, the solution achieving less average backup path latency is a better solution when $s = 2$.

4.3. Perturbation

Perturbation reassigns some of the switches to other controllers for each node subset as shown in Algorithm 5. Perturbation consists of two phases: destruction and remapping. In the destruction phase, a percentage α $(0 < \alpha < 1)$ of the switches will be removed from their mapped controller for each node subset. Assuming switch i is originally mapped to controller c_k, the association between switch i and controller c_k is removed with the probability of ρ_i defined by Eq. (15). The switch with a large delay to its assigned controller is likely to be unassigned.

$$\rho_i = \frac{l_{i,k}^p}{\sum_{j \in X_k'} l_{j,k}^p}. \tag{15}$$

The remapping phase first sorts the removed nodes in the non-ascending order of the delay between the nodes and the originally assigned controllers and then maps each of the removed nodes to the closest (least primary paths delay) controller in turn under the controller processing capacity constraint.

4.4. Controller relocation

Controller relocation attempts to increase the performance of the feasible solutions by moving a controller to a different position in the same node subset as illustrated in Algorithm 6. For controller c_k and corresponding subset X_k' in feasible solution X', the benefit of moving controller to node q is defined by Eqs. (16) and (17).

$$\pi_3^1(k, q) = \sum_{i \in X_k'} (l_{i,k}^p - l_{i,q}^p). \tag{16}$$

$$\pi_3^2(k, q) = \sum_{i \in X_k'} (l_{i,k}^b - l_{i,q}^b). \tag{17}$$

where $\pi_3^s(k, q) > 0$ $(s \in \{1, 2\})$ indicates that moving the position of controller c_k to node q will reduce the node-to-controller delay. For each node subset in the feasible solution, Algorithm 6 attempts to find a better solution by checking all the different node positions for the controller in the corresponding subset. If a better position for the controller is found, the controller is moved to the found place, and Algorithm 6 reassigns all the switches in the subset to the controller at the new location. The algorithm then decides whether to accept the new solution as a non-dominated solution to the problem and performs *Update* operation on the Pareto frontier.

4.5. Shake

Shake shown in Algorithm 7 destroys the current solution significantly so that the solution can jump to a solution space far away. Shake attempts to find better controllers for the nodes with the largest node-to-controller communication delay in each subset. We denote τ_i as the controller to which switch i is mapped and $\ell(X')$ as the set of switches, each having the largest primary path delay to the associated controller in its corresponding subset. For each switch $i \in \ell(X')$, Shake destructs η number of the subsets in the current solution. We define $W^i = \{X_{k_1}', X_{k_2}', \ldots, X_{k_\eta}'\}$, where $X_{k_1}', X_{k_2}', \ldots, X_{k_\eta}'$ are the η subsets with the smallest primary path delay between switch i and the corresponding controllers in the η subsets, assuming k_i is the index of the controller in subset X_{k_i}'. The η subsets are sorted, with X_{k_1}' as the subset having the least primary path delay from the subset controller to switch $i \in W^i$. We create new topology $G' = \{V', E', W'\}$, where V' consists of all nodes $i \in W^i$ and $E' = \{(i, j) |$ every $i, j \in V'\}$, $W' = \{w_{i,j} | w_{i,j}$ is the length of path $p_{i,j}$ in original topology G, $i, j \in V'\}$. For each switch i, if $\tau_i = k_1$, we destroy all the η subsets in W^i, and reassign all the nodes in W^i with the same process in the initial solution construction. The solution might be destroyed greatly by remapping these nodes.

If there is no node $i' \in \ell(X')$ mapped to the closest controller $(\tau_{i'} = k_1)$, steps (1–10) do not destroy the current solution. Algorithm 7 then reconstructs all the subsets of the current solution. The reconstruction method is slightly different from the initial solution construction process in terms of the selection of the K potential controller positions in the subset partition. We calculate $\phi(i)$ for each node $i \in V$ as

$$\phi(i) = \frac{\sum_{i' \in H^i} r_{i'}}{\sum_{j \in V} r_j}, \tag{18}$$

where $\phi(i)$ is the probability of selecting node i as a potential controller position.

A large amount of the requests from all the nodes that can reach node i in two hops results in a greater value of $\phi(i)$. Each node is selected as a potential controller position with probability $\phi(i)$ $(i \in V)$. We select K potential controller positions and form K node subsets with each having only one node, i.e., the node at the selected position. Algorithm 7 then assigns each node i to subset S_k which incurs the minimum primary path delay from node i to the kth potential controller location. Note that if the value of $\phi(i)$ $(\forall i \in V)$ remains unchanged, the algorithm will potentially generate a few same node subset partitions, since node i with a high value of $\phi(i)$ is always likely to be selected

Algorithm 3 *Construction*

Input: Network topology $G = (V, E)$, Pareto optimal solution set $\mathcal{X}$, the number of requests of the switches, the number of controllers K, subset division S'.
Output: Feasible solution X'.

for $1 \le k \le K$ **do**
  $X_k = \emptyset$;
end for
for $1 \le k \le K$ **do**
  $k^* = \underset{i \in S_k}{\operatorname{argmax}} \sum_{j \in H^i} r_j$;
  $c_k = c_{k^*}$;
end for
for $1 \le k \le K$ **do**
  Sort all the nodes in subset S_k by the non-ascending order of the total number of requests from the nodes;
  for each $i \in S_k$ **do**
    if controller c_k has enough capacity to serve the requests from node i **then**
      $X_k = X_k \cup \{i\}$;
    end if
  end for
end for
if there are unassigned nodes **then**
  Assign each of the unassigned nodes to the closest controller with enough capacity to accommodate the requests from the node;
end if
Update($\mathcal{X}$);
return X'.

as a potential controller position. To achieve the diversity of node subset partitions, each $\phi(i)$ is considered as a global variable and it is initialized by Eq. (18) at the beginning of the proposed algorithm RALO. The value of $\phi(i)$ is decreased by step-length ν, after node i is selected as a potential controller position. $\phi(i)$ is restored to its initial value by Eq. (18), when $\phi(i)$ is decreased to a value lower than a threshold $\mathcal{T}$. In this paper, we set both step-length ν and threshold $\mathcal{T}$ as $\frac{1}{2}\min_{i \in V} \phi(i)$.

5. Performance evaluation

In this section, we evaluate the performance of the proposed controller placement algorithm RALO. We also investigate the impact of important components on the performance of the proposed algorithm.

5.1. Simulation setup

We evaluate the performance of the proposed algorithm against the optimal solution *OPT* [31], algorithm LARC [30], algorithm Survivor [25], and algorithm PSA [24]. We obtain the optimal solution on each objective in the controller placement problem defined in Section 3 via CPLEX 12.8.0 by ignoring the other objective. Algorithm LARC introduces the accumulated delay by integrating the two sub-objectives of the primary path delay and backup path latency into one objective; that is, algorithm LARC minimizes the weighted sum of the primary and the backup path delay as $l_{i,j} = \lambda_1 l^p_{i,j} + \lambda_2 l^b_{i,j}$ ($\lambda_1 + \lambda_2 = 1$, $0 \le \lambda_1, \lambda_1 \le 1$). Algorithm LARC places each controller by searching the location that incurs the least path cost between each unassigned switch and the controller. Algorithm Survivor places the controllers by solving the linear programming problem to maximize the average number of disjoint paths between each switch and its controller. Algorithm PSA used in extended POCO [24] is a meta-heuristic multi-objective optimization algorithm based on simulated annealing; the algorithm drops the temperature from an initial value to 1 and searches the solution for many times at each temperature during the dropping. In this paper, the initial temperature and the number of search times at each temperature are set as 50 and 90 for algorithm PSA, respectively [24]. Algorithm RALO deals with the multi-objective problem and provides a Pareto optimal solution set which allows network administrators with flexible choices of the solutions. For comparison of these two algorithms, we select one solution from the Pareto optimal solution set obtained by Algorithm RALO and calculate the accumulated (weighted) delay in the same way as algorithm LARC.

Table 2
The parameters of real network topologies.

	Alilene	Arnes	ATT	Darkstrand	Internet2	Savvis	Spiralight	Xspedius
$\mathcal{N}$	11	34	25	28	34	19	15	34
$\mathcal{M}$	14	46	56	31	42	20	16	49
$\mathcal{R}$	[3, 8]	[8, 13]	[6, 11]	[7, 12]	[9, 14]	[4, 9]	[4, 9]	[8, 13]

Table 3
The parameters of Generated networks I and Generated networks II.

		1	2	3	4	5	6	7	8	9	10
I	$\mathcal{N}$	50	50	50	50	50	50	50	50	50	50
	$\mathcal{M}$	100	100	150	150	200	200	250	250	300	300
II	$\mathcal{N}$	50	50	50	50	50	50	50	50	50	50
	$\mathcal{K}$	2	2	2.5	2.5	3	3	3.5	3.5	4	4

The network topologies used in the simulations are 8 real network topologies from Internet topology zoo [32], as well as two kinds of network topologies (Generated networks I and Generated networks II) generated by Stanford Network Analysis Platform (SNAP) [33]. The parameters of the real networks are shown in Table 2, where $\mathcal{R}$ is the range of the number of controllers, i.e. $[K_{min}, K_{max}]$, and $\mathcal{N}$ and $\mathcal{M}$ denote the number of nodes and edges, respectively. Generated networks I is a set of network topologies which conform to the ER (Erdos–Renyi) random model, while Generated networks II is a set of network topologies which conform to the small-world model. Each kind of generated networks includes 10 network topologies. The parameters of these generated networks are shown in Table 3, where $\mathcal{K}$ is the average number of the nearest neighbors to which each node is connected.

All the controllers have identical computing capacity of 1000 kilo-requests/s, and each switch generates the requests with a rate of 200 kilo-requests/s. We use the geographical distance between two locations as an approximation of latency [6]. The latency on each edge

Algorithm 4 *VNS*

Input: Network topology $G = (V, E)$, Pareto optimal solution set $\mathcal{X}$, the number of requests of the switches, the number of controllers K, feasible solution X' and parameter β.
Output: New feasible solution X''.

$h = 1$;
$s = 1$;
while $s \leq 2$ **do**
  while $h \leq 2$ **do**
    $X'' = X'$;
    for each $i \in \mathcal{L}_s(X'', \beta)$ **do**
      if $h = 1$ **then**
        $k^* = \underset{q}{\operatorname{argmax}}\, \pi_1^s(i, k, q)$;
        if $\pi_1^s(i, k, k^*) > 0$ & $u_k^* \geq r_i$ **then**
          remap(i, k, k^*);
          Update$(\mathcal{X})$;
        end if
      end if
      if $h = 2$ **then**
        for each $i \in V$ **do**
          $j^* = \underset{j}{\operatorname{argmax}}\, \pi_2^s(i, j)$;
          if $\pi_2^s(i, j^*, k, q) > 0$ & $u_k + r_i \geq r_{j^*}$ & $u_q + r_{j^*} \geq r_i$ **then**
            swap(i, j^*);
            Update$(\mathcal{X})$;
          end if
        end for
      end if
    end for
    if X'' *is better than* X' **then**
      $X' = X''$;
      $h = 1$;
    else
      $h = h + 1$;
    end if
  end while
  $s = s + 1$;
end while
return X''.

in the generated networks is randomly generated. In the simulations, the number of iterations, r_{max}, during the algorithm execution is 200, and the parameters are set as $\alpha = 0.3$, $\beta = 3$, $\eta = \lceil lnK \rceil + 1$. For generated networks I and generated networks II, we take the average of the running results on all the network topologies as the simulation results for each kind of generated networks.

5.2. Performance evaluation of the proposed algorithm

In this section, we define *GAP* via Eq. (19) to indicate the difference between the result obtained by an algorithm and the optimal solution OPT.

$$GAP = |\frac{R-OPT}{OPT}|. \tag{19}$$

where R represents the result obtained by an algorithm. The smaller the value of *GAP*, the better the result obtained by the algorithm.

5.2.1. Average latency of primary paths

Table 4 lists the average latency of primary paths obtained by different algorithms of RALO, LARC, Survivor, PSA, and the optimal solution versus the different number of controllers in the real network topologies. K_{min} and K_{max} in the column of Network represent the range of the number of controllers in the network. The weights of the primary and the backup path latencies with algorithm LARC are set as $\lambda_1 = 1$ and $\lambda_2 = 0$, respectively. That is, algorithm LARC only cares about the primary path latency and ignores the backup path delay. In this way, algorithm LARC can achieve the best primary path delay performance. The smallest *GAP* obtained by algorithms LARC, RALO, and PSA is 0, 0, and 0.0015, respectively. That is, both RALO and PSA can obtain the optimal solution at least once. For example, both RALO and PSA find the optimal result on network Spiralight when the number of controllers is 9. The largest *GAP* obtained by algorithms LARC, RALO, PSA, and Survivor is 0.3560, 0.1211, 0.3560, and 6.0447, respectively. In conclusion, algorithm RALO outperforms algorithms LARC, PSA, and Survivor in terms of primary paths latency.

Figs. 1(a)–(b) plot the average latency of primary paths with different numbers of controllers for RALO, LARC, PSA, Survivor, and OPT on generated networks I and generated networks II, respectively. Algorithm RALO achieves better performance than algorithms LARC, PSA, and Survivor on all these generated networks. In general, the primary path latency of all the algorithms decreases as the number of controllers increases, since each switch can be potentially mapped to a closer controller with more controllers in the network. Algorithm RALO outperforms algorithm LARC by up to 2.7% and 3.8% on generated networks I and generated networks II, respectively. The average primary latency obtained by algorithm RALO is better than that obtained by algorithm PSA by 1.8%–2.7% on generated networks I and 18%–25% on generated networks II, respectively. Algorithm RALO searches for a large solution space to find a wide range of good solutions, while algorithm LARC minimizes the weighted sum of the primary and the

Algorithm 5 *Perturbation*

Input: Network topology $G = (V, E)$, the number of requests of the switches, the number of controllers K, the controller set C, feasible solution X' and parameter α.
Output: New feasible solution X''.

$X'' = X'$;
for each $c_k \in C$ **do**
  $W_k = \emptyset$;
  for each $i \in X'_k$ **do**
    calculate $\rho_i = \frac{l^p_{i,k}}{\sum_{j \in X'_k} l^p_{j,k}}$;
  end for
  while $\frac{|W_k|}{|X'_k|-1} < \alpha$ **do**
    Choose $i \in X'_k$ randomly to be removed using probability ρ_i;
    if i is chosen to be removed **then**
      $W_k = W_k \cup \{i\}$;
    end if
  end while
end for
$W = \{W_1, W_2, \ldots, W_K\}$;
Sort all the switches in W in the non-ascending order of the delay from the switches to the originally mapped controllers;
for each $i \in W$ **do**
  Map switch i to the nearest controller c_k with sufficient processing capacity;
  $X''_k = X''_k \cup \{i\}$;
end for
return X''.

Algorithm 6 *Relocation*

Input: Network topology $G = (V, E)$, Pareto optimal solution set $\mathcal{X}$, the number of requests of the switches, the number of controllers K, the controller set C, feasible solution X'.
Output: New feasible solution X''.

$X'' = X'$, $s = 1$;
while $s \leq 2$ **do**
  for each $c_k \in C$ **do**
    for each $q \in X''_k$ **do**
      if $\pi^s_3(k, q) > 0$ **then**
        Move controller c_k to node q;
        Update($\mathcal{X}$);
      end if
    end for
  end for
  $s = s + 1$;
end while
return X''.

backup path delay. The worst average primary latency performance is obtained by algorithm Survivor. Survivor aims to maximize the number of disjoint paths between the switches and the associated controllers, and Survivor ignores the latency during controller placements.

5.2.2. Average latency of backup paths

Table 5 shows the average latency of backup paths obtained by the algorithms and the optimal solution with a different number of controllers in the real network topologies. The weights of the primary and the backup path latencies with algorithm LARC are set as $\lambda_1 = 0$ and $\lambda_2 = 1$, respectively. That is, algorithm LARC only optimizes the backup path latency by ignoring the primary path delay. In this way, algorithm LARC can achieve the best backup path delay performance within the capability of LARC. The *GAP* obtained by RALO, PSA, LARC, and Survivor varies from 0 to 0.5143, from 0 to 0.986, from 0.0008 to 3.4644, and from 0.2875 to 7.4776, respectively. In general, algorithm RALO performs better than algorithms PSA, LARC, and Survivor in terms of backup path latency.

Figs. 2(a)–(b) demonstrate the average latency of backup paths with algorithms RALO, LARC, PSA, Survivor, and the optimal solution with different numbers of controllers in generated networks I and generated networks II, respectively. In general, the average backup path latency decreases with the increase of the number of controllers. More controllers enable wider controller distribution in the network so that the backup latency is reduced. Algorithm RALO always obtains better performance than algorithms LARC, PSA, and Survivor on all the generated networks, since algorithm RALO finds the controller positions after traversing a large solution space. Particularly, the performance improvement of algorithm RALO over algorithms LARC, PSA, and Survivor reaches up to 12.9%, 8.1%, and 36.1%, respectively, in generated networks I.

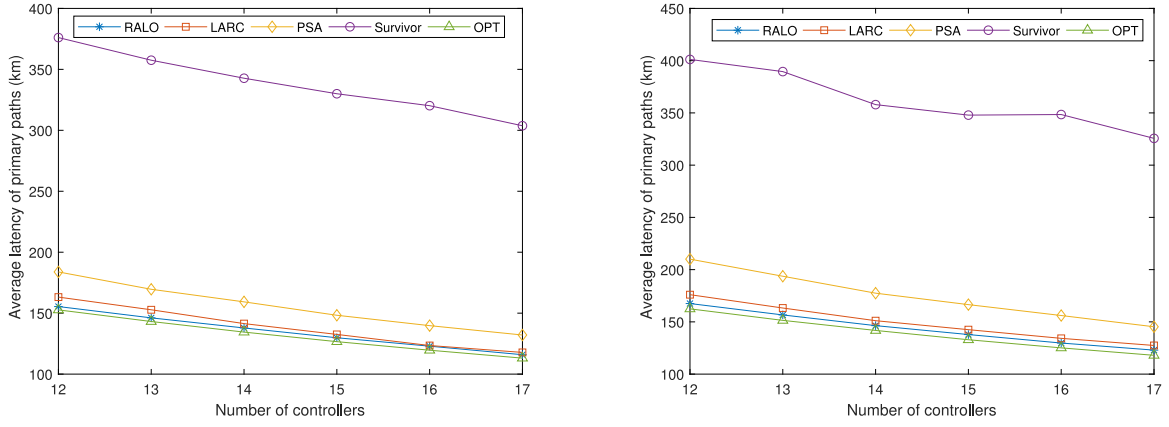
5.2.3. Average accumulated latency with different numbers of controllers

In this section, we evaluate the performance of the algorithms in terms of average accumulated latency. The accumulated latency is the

Algorithm 7 *Shake*

Input: Network topology $G = (V, E)$, the number of requests of the switches, the number of controllers K, feasible solution X' and parameter η.
Output: New feasible solution X''.

for all $i \in \ell(X')$ **do**
  Calculate W^i;
  Create G';
  if $k_1 = \tau_i$ **then**
    $X' = X' \setminus W^i$;
    $S' = \text{Division}(G', \eta)$;
    $X'' = \text{Construction}(G', S', \eta)$;
    $X'' = X' \cup X''$;
  end if
end for
if there is no $k_1 = \tau_{i'}$ for all $i' \in \ell(X')$ **then**
  $C' = \varnothing$;
  while $|C'| < K$ **do**
    Choose $i \in V$ using probability $\phi(i) = \frac{\sum_{i' \in H^i} r_{i'}}{\sum_{j \in V} r_j}$;
    $C' = C' \cup \{i\}$;
    $\phi(i) = \phi(i) - \nu$;
    if $\phi(i) < \mathcal{T}$ **then**
      $\phi(i) = \frac{\sum_{i' \in H^i} r_{i'}}{\sum_{j \in V} r_j}$;
    end if
  end while
  for $1 \le k \le K$ **do**
    $S_k = \varnothing$;
  end for
  for $i \in V$ **do**
    $k = \underset{k' \in C}{\text{argmin}}\, l^p_{i,k'}$;
    $S_k = S_k \cup \{i\}$;
  end for
  $X'' = \text{Construction}(G, S, K)$;
end if
return X''.



(a) Generated networks I (b) Generated networks II

Fig. 1. The average latency of primary paths on generated networks.

weighted sum of primary and back path latency, and the weights of the primary and the backup path latency are set as $\lambda_1 = 0.8$ and $\lambda_2 = 0.2$, respectively.

Table 6 lists the accumulated latency obtained by all the algorithms and the optimal solution with different numbers of controllers in the real network topologies. The average *GAP* obtained by algorithms RALO, LARC, PSA, and Survivor is 0.0201, 0.0916, 0.0872, and 1.3218, respectively. The average *GAP* of algorithm RALO is the smallest among the four algorithms; that is, the accumulated latency obtained by algorithm RALO is smaller than that obtained by algorithms LARC, PSA, and Survivor.

Figs. 3(a)–(b) illustrate the average accumulated latency by varying the number of controllers in the networks of generated networks I and generated networks II, respectively. In general, the accumulated

Table 4
The average latency (km) of primary paths on real network topologies.

Network	Algorithm	K_{min}	$K_{min}+1$	$K_{min}+2$	$K_{min}+3$	$K_{min}+4$	K_{max}	Best *GAP*	Worst *GAP*
	LARC	682.14	469.20	390.06	243.40	166.15	116.44	0.0256	0.1749
Abilene	RALO	662.55	441.46	332.00	228.45	162.00	99.50	0.0000	0.1211
($K_{min}=3$)	PSA	701.81	441.45	346.63	228.45	162.00	99.50	0.0000	0.1875
($K_{max}=8$)	Survivor	1281.46	850.00	1191.09	539.55	619.36	294.18	0.9274	2.8232
	OPT	591.00	441.00	332.00	228.00	162.00	99.50		
	LARC	18.19	14.73	13.54	12.14	11.55	10.34	0.0412	0.1192
Arnes	RALO	17.47	13.85	12.70	11.47	10.32	9.44	0.0000	0.0160
($K_{min}=8$)	PSA	19.47	15.64	13.44	12.70	11.58	10.14	0.0742	0.1292
($K_{max}=13$)	Survivor	48.82	38.88	40.29	35.29	41.29	39.50	1.7947	3.1843
	OPT	17.47	13.85	12.50	11.47	10.32	9.44		
	LARC	511.92	393.30	280.16	237.08	214.31	193.41	0.0135	0.3497
ATT	RALO	379.28	308.20	259.76	233.92	210.04	186.96	0.0000	0.0001
($K_{min}=6$)	PSA	442.92	402.04	327.20	297.20	263.16	216.12	0.1560	0.3045
($K_{max}=11$)	Survivor	1322.48	1227.6	1225.96	1277.28	961.8	595.68	2.1861	4.4603
	OPT	379.28	308.20	259.76	233.92	210.01	186.96		
	LARC	396.21	348.16	272.36	258.76	238.04	220.31	0.0187	0.1358
Darkstrand	RALO	362.07	306.53	267.57	243.57	223.07	202.75	0.0000	0.0008
($K_{min}=7$)	PSA	449.03	377.64	316.60	271.25	241.60	232.64	0.0831	0.2402
($K_{max}=12$)	Survivor	1409.04	1651.07	1161.61	1202.50	1149.82	1028.29	2.8916	4.3863
	OPT	362.07	306.53	267.35	243.57	223.07	202.75		
	LARC	330.85	291.38	252.92	234.77	215.40	200.60	0.0070	0.0581
Internet2	RALO	312.67	278.58	251.17	231.82	214.58	195.26	0.0000	0.0279
($K_{min}=9$)	PSA	386.47	345.00	293.05	267.17	245.79	220.79	0.1525	0.2384
($K_{max}=14$)	Survivor	1347.41	1218.65	970.82	1063.21	1204.97	1038.47	2.8652	4.7720
	OPT	312.67	278.58	251.17	231.82	208.76	190.11		
	LARC	493.12	376.89	272.73	216.29	195.89	167.01	0.0015	0.1338
Savvis	RALO	465.78	332.42	272.31	215.94	185.73	154.31	0.0000	0.0066
($K_{min}=4$)	PSA	564.89	354.42	292.68	222.73	186.52	154.31	0.0000	0.2128
($K_{max}=9$)	Survivor	1404.68	1385.16	1311.95	965.84	1299.90	1015.74	2.0158	6.0447
	OPT	465.78	332.42	272.31	215.94	184.52	154.31		
	LARC	57.25	57.35	34.74	34.17	20.75	16.79	0.0079	0.3560
Spiralight	RALO	56.80	43.80	34.46	25.20	19.46	15.93	0.0000	0.0304
($K_{min}=4$)	PSA	56.80	44.93	34.46	25.20	19.46	15.93	0.0000	0.0304
($K_{max}=9$)	Survivor	182.13	191.47	157.73	164.33	94.53	43.47	1.8116	5.5212
	OPT	56.80	43.80	34.46	25.20	19.46	15.46		
	LARC	395.39	321.42	256.12	235.46	224.45	193.98	0.0210	0.1931
Xspedius	RALO	331.41	286.55	250.85	223.29	202.00	180.73	0.0000	0.0000
($K_{min}=8$)	PSA	432.64	326.08	281.73	279.61	238.79	217.32	0.1231	0.3055
($K_{max}=13$)	Survivor	997.09	1030.03	959.82	800.41	877.94	772.79	2.0086	3.3462
	OPT	331.41	286.55	250.85	223.29	202.00	180.73		

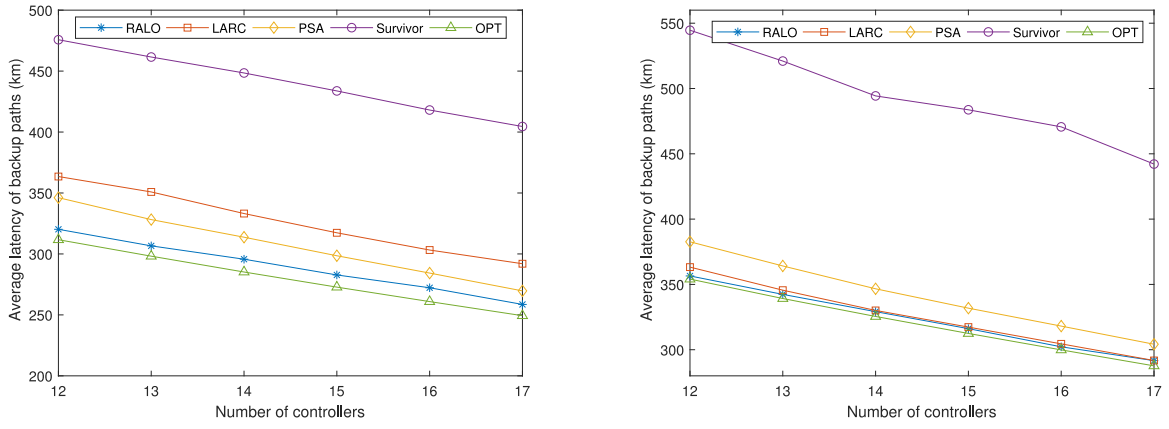


Fig. 2. The average latency of backup paths on generated networks.

latency resulted from all the algorithms decreases with the increase of the number of controllers, since the switches can be mapped to closer controllers with more controllers available in the network. Algorithm RALO achieves better performance than algorithms LARC, PSA, and Survivor on all the generated networks, since algorithm RALO obtains better results than algorithms LARC and PSA in both primary and backup path delay, and algorithm RALO can provide multiple solutions for the multi-objective problem. Particularly, the performance improvement of algorithm RALO on algorithms LARC and PSA reaches up to 4.3% and 16.3%, respectively, in generated networks II. Algorithm Survivor has the worst performance among all the algorithms because

Table 5
The average latency (km) of backup paths on real network topologies.

Network	Algorithm	K_{min}	$K_{min}+1$	$K_{min}+2$	$K_{min}+2$	$K_{min}+3$	K_{max}	Best *GAP*	Worst *GAP*
	LARC	1803.38	1514.50	1227.05	1004.38	692.62	519.14	0.0008	0.1093
Abilene	RALO	1756.82	1444.36	1145.82	905.45	692.09	494.72	0.0000	0.0327
($K_{min}=3$)	PSA	1736.27	1428.82	1145.82	932.45	692.09	494.72	0.0000	0.0298
($K_{max}=8$)	Survivor	2251.43	1907.55	1752.99	1165.73	1071.73	721.64	0.2875	0.5485
	OPT	1736.27	1398.63	1145.81	905.45	692.09	494.72		
	LARC	52.49	58.62	47.91	43.23	41.34	35.17	0.4416	0.9686
Arnes	RALO	45.09	37.60	32.40	30.58	28.74	24.80	0.1944	0.3686
($K_{min}=8$)	PSA	45.53	39.01	34.90	30.83	28.42	22.88	0.2392	0.3533
($K_{max}=13$)	Survivor	88.16	86.42	81.18	78.76	80.53	65.17	1.4213	2.8347
	OPT	36.41	31.48	27.00	23.95	21.00	18.24		
	LARC	758.54	675.79	634.44	557.37	515.42	450.86	0.0341	0.2026
ATT	RALO	733.56	654.64	592.16	531.56	478.08	428.60	0.0000	0.1154
($K_{min}=6$)	PSA	769.46	697.08	613.20	567.28	500.00	441.16	0.0489	0.1866
($K_{max}=11$)	Survivor	1497.78	1375.32	1381.88	1357.59	1124.46	727.84	0.8921	1.8397
	OPT	733.56	592.16	531.56	478.08	428.60	384.68		
	LARC	1831.90	1669.37	1543.34	1445.88	1327.90	1214.90	0.0266	0.0500
Darkstrand	RALO	1797.23	1651.15	1551.76	1439.26	1329.01	1211.10	0.0234	0.0336
($K_{min}=7$)	PSA	1798.04	1680.02	1564.51	1440.91	1334.04	1218.23	0.0294	0.0430
($K_{max}=12$)	Survivor	2651.29	2809.00	2463.57	2160.75	2386.47	2199.24	0.5196	0.8584
	OPT	1744.72	1610.73	1501.26	1394.15	1288.07	1183.43		
	LARC	1258.47	1170.27	1105.92	990.49	890.07	831.31	0.6289	0.7031
Internet2	RALO	1041.27	824.06	774.77	688.55	622.41	580.24	0.1502	0.3477
($K_{min}=9$)	PSA	1048.12	830.59	783.85	690.07	629.94	582.82	0.1580	0.3566
($K_{max}=14$)	Survivor	2073.02	1913.66	1692.98	1701.36	1763.95	1651.11	1.5930	2.3827
	OPT	772.61	711.61	652.91	595.94	541.12	488.11		
	LARC	2104.69	2059.94	1822.42	1674.15	1487.17	1299.85	0.8424	3.4644
Savvis	RALO	1214.16	945.68	757.42	616.73	440.89	440.89	0.0000	0.5143
($K_{min}=4$)	PSA	1316.84	958.63	772.10	616.73	454.89	440.89	0.0230	0.5143
($K_{max}=9$)	Survivor	3686.80	3311.53	3099.17	2668.16	2580.70	2468.35	2.2274	7.4776
	OPT	1142.36	930.57	754.73	590.84	440.89	291.16		
	LARC	314.68	277.37	235.30	199.02	161.62	124.32	0.0088	0.0330
Spiralight	RALO	329.03	280.36	236.30	200.10	163.50	129.20	0.0228	0.0665
($K_{min}=4$)	PSA	332.46	285.26	238.46	200.60	165.86	128.40	0.0322	0.0777
($K_{max}=9$)	Survivor	441.07	363.33	414.80	253.67	298.13	194.67	0.3105	0.8874
	OPT	308.50	268.50	231.03	193.56	157.96	123.23		
	LARC	976.37	874.66	801.62	740.75	684.99	632.74	0.3158	0.3488
Xspedius	RALO	801.96	700.98	673.01	602.50	567.31	516.52	0.0545	0.1102
($K_{min}=8$)	PSA	834.27	757.01	678.28	627.31	574.89	515.04	0.0954	0.1525
($K_{max}=13$)	Survivor	1422.19	1462.13	1278.64	1183.30	1288.68	1065.51	0.9647	1.5196
	OPT	723.88	664.73	606.23	556.85	511.47	470.17		

Survivor performs poorly on both primary path delay and backup path latency.

5.2.4. Average accumulated latency with different backup path latency weights

Figs. 4(a)–(d) show the average accumulated latency of RALO, LARC, PSA, Survivor, and OPT by varying λ_2, the weight of backup path latency, given the numbers of controllers are 7, 10, 13, and 13 in ATT, Internet2, generated networks I, and generated networks II, respectively.

Algorithm RALO always achieves better results than algorithm LARC with different backup path latency weights on all the four kinds of networks. The average accumulated latency increases with the growth of the weights of backup path latency since the backup path latency is larger than the primary path latency. The average accumulated latency obtained by algorithm RALO is better than that obtained by algorithm LARC by 0.4%–2.4% in generated networks I and by 0.3%–3.8% for Generated networks II, respectively. The performance improvement of algorithm Algorithm RALO on algorithm LARC reaches up to 6% for network ATT and 12% for network Internet2. Algorithm RALO outperforms algorithm PSA by up to 18.5%, 7.6%, 8.2%, and 17.3% on the networks of ATT, Internet2, generated networks I and generated networks II, respectively. The accumulated latency of algorithm Survivor is worse than that of algorithms RALO, LARC, and PSA. The results obtained by algorithm RALO are worse than OPT from 0 to 12.5% on all the four kinds of networks.

Hereinabove, we analyze the primary path latency, the backup path latency, and the accumulated path latency with different numbers of controllers, and study the accumulated path latency by varying the weights of backup path latency. The simulation results show that the proposed algorithm RALO achieves better performance than algorithms LARC, PSA, and Survivor in all cases.

5.2.5. Uniformity and spread performance of Pareto optimal solutions

We analyze the uniformity and spread performance of the obtained Pareto optimal solutions. We only compare RALO with PSA, since only these two algorithms produce Pareto optimal solution sets. We introduce two performance metrics for measuring the performance of the algorithms, i.e. Spacing [34] and Maximum Spread [35]. Spacing metric SP evaluates the uniformity of the obtained solutions, and SP is defined as

$$SP=\sqrt{\frac{\sum_{X\in\mathcal{X}}\left(\overline{d}-d_X\right)^2}{|\mathcal{X}|-1}} \tag{20}$$

where

$$d_X=\min_{X'\in\mathcal{X},X'\neq X}\left\{\sum_{g=1}^{M}\left|f_g(X)-f_g\left(X'\right)\right|\right\} \tag{21}$$

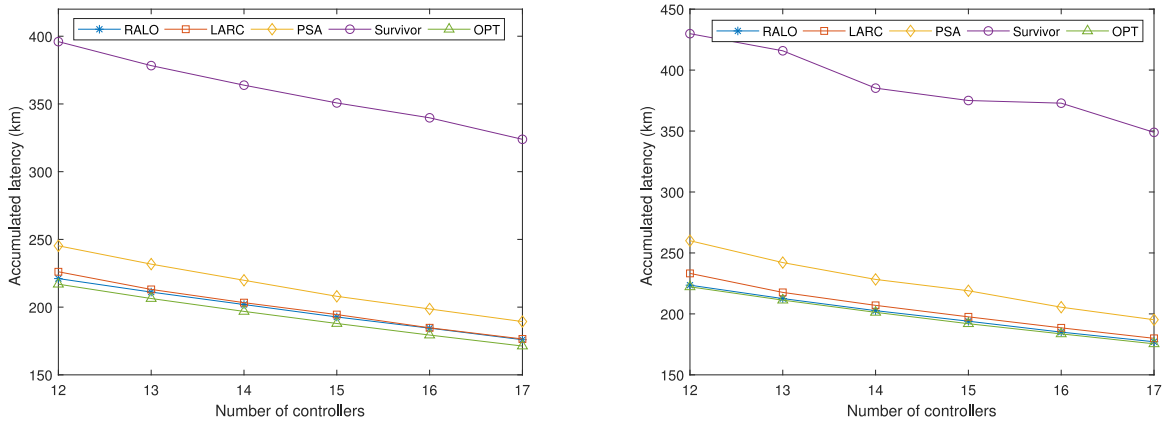
and

$$\overline{d}=\frac{\sum_{X\in\mathcal{X}}d_X}{|\mathcal{X}|} \tag{22}$$

In Eqs. (20)–(22), M is the number of optimized objectives, and $f_g(X)$ represents the value of solution X on objective g. The smaller

Table 6
The accumulated latency (km) on real network topologies.

Network	Algorithm	K_{min}	$K_{min}+1$	$K_{min}+2$	$K_{min}+3$	$K_{min}+4$	K_{max}	Best *GAP*	Worst *GAP*
	LARC	1004.48	796.73	691.38	494.48	372.11	257.55	0.0008	0.0903
Abilene	RALO	978.56	794.16	634.14	494.09	354.80	249.65	0.0000	0.0234
($K_{min}=3$)	PSA	1032.68	811.67	651.16	494.09	388.94	257.30	0.0000	0.0962
($K_{max}=8$)	Survivor	1475.45	1061.51	1303.47	664.78	709.84	379.67	0.3366	1.0555
	OPT	956.20	794.16	634.14	494.09	354.80	249.65		
	LARC	31.35	28.69	25.87	23.60	21.46	20.01	0.0885	0.1743
Arnes	RALO	28.97	26.98	24.02	21.84	20.44	18.14	0.0059	0.0702
($K_{min}=8$)	PSA	31.64	27.46	25.48	23.40	20.86	19.76	0.0627	0.1596
($K_{max}=13$)	Survivor	56.69	48.39	48.47	43.99	49.14	44.63	0.8727	1.6193
	OPT	28.80	25.84	23.57	21.31	19.10	17.04		
	LARC	617.82	451.15	376.26	337.19	301.49	265.49	0.0116	0.2811
ATT	RALO	482.24	425.85	371.96	332.38	293.35	262.36	0.0000	0.0138
($K_{min}=6$)	PSA	517.32	465.60	428.89	370.71	332.36	298.60	0.0727	0.1531
($K_{max}=11$)	Survivor	1357.54	1257.14	1257.14	1293.34	994.33	622.11	1.3712	2.9258
	OPT	482.24	420.07	371.96	329.45	293.35	262.36		
	LARC	889.99	825.86	716.89	665.56	614.11	571.51	0.0012	0.0618
Darkstrand	RALO	843.01	781.42	723.59	673.86	624.52	571.05	0.0016	0.0182
($K_{min}=7$)	PSA	921.16	823.76	762.66	703.48	637.59	601.14	0.0395	0.0944
($K_{max}=12$)	Survivor	1 1657.49	1882.66	1422.00	1394.15	1397.15	1262.48	0.9692	1.4206
	OPT	841.70	777.77	716.04	664.65	613.36	563.49		
	LARC	628.08	584.99	550.25	495.53	463.02	442.08	0.0779	0.1192
Internet2	RALO	609.61	569.55	525.59	486.53	455.59	425.41	0.0583	0.0800
($K_{min}=9$)	PSA	628.70	581.52	531.52	501.07	461.68	425.63	0.0727	0.1069
($K_{max}=14$)	Survivor	1492.53	1357.65	1115.26	1190.84	1316.77	1161.00	1.2684	2.0817
	OPT	567.98	527.36	491.64	459.72	427.29	396.77		
	LARC	1084.46	922.22	831.68	739.85	668.25	569.99	0.2419	0.3714
Savvis	RALO	844.54	742.58	673.41	585.62	501.47	432.29	0.0000	0.0477
($K_{min}=4$)	PSA	1052.68	869.42	742.61	644.72	585.53	451.43	0.0742	0.2465
($K_{max}=9$)	Survivor	1861.11	1770.43	1669.39	1306.31	1556.06	1306.26	1.2037	2.1934
	OPT	844.54	742.58	649.41	558.97	487.27	420.24		
	LARC	152.85	127.34	108.73	91.96	76.45	58.93	0.0058	0.0558
Spiralight	RALO	148.21	127.82	108.13	89.93	72.98	56.90	0.0000	0.0117
($K_{min}=4$)	PSA	148.45	128.93	109.69	90.10	73.02	57.46	0.0071	0.0220
($K_{max}=9$)	Survivor 233.92	225.84	209.15	182.20	135.25	73.71	0.2954	1.0497	
	OPT	147.41	126.61	107.33	88.89	72.41	56.90		
	LARC	563.75	492.59	436.90	406.59	369.70	346.66	0.0099	0.0969
Xspedius	RALO	516.54	463.76	425.97	395.75	367.07	341.28	0.0014	0.0088
($K_{min}=8$)	PSA	586.60	493.35	467.23	452.79	404.74	359.98	0.0641	0.1457
($K_{max}=13$)	Survivor	1082.11	1116.45	1023.59	876.99	960.08916	831.33768	1.1054	1.6226
	OPT	513.97	462.67	424.88	395.20	366.08	338.31		



(a) Generated networks I (b) Generated networks II

Fig. 3. The accumulated latency on Generated networks.

the value of SP, the better uniformity of the solution set. Metric MS measures the spread performance of the solution set, and MS is defined as

$$MS = \sqrt{\sum_{g=1}^{M}\left(\max_{X\in\mathcal{X}} f_g(X) - \min_{X\in\mathcal{X}} f_g(X)\right)^2} \tag{23}$$

The larger the value of MS, the better spread performance of the solution set.

Fig. 5 illustrates the Pareto optimal solutions obtained by algorithms RALO and PSA on 8 real network topologies with 7 controllers. To better evaluate the performance of the proposed algorithm, the SP and

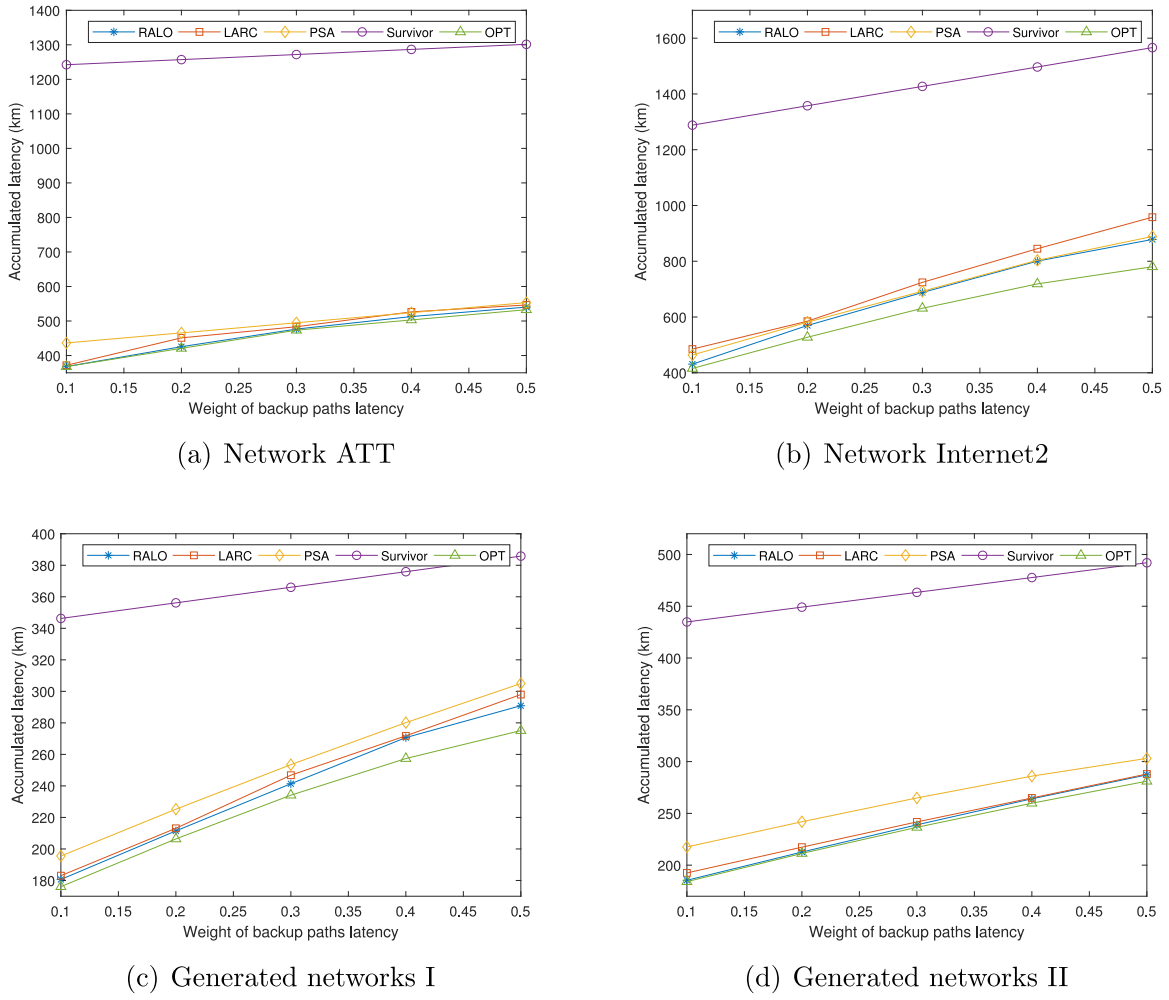


Fig. 4. The average accumulated latency by varying the latency weights of backup paths.

MS values of the Pareto optimal solution sets obtained by algorithms RALO and PSA are listed in Table 7.

For the metric of spacing, the SP of algorithm RALO is 20.6%–83.3% smaller than that of algorithm PSA. For the metric of maximum spread, the MS of algorithm RALO is 0%-74.3% larger than that of algorithm PSA. Algorithm RALO is better than algorithm PSA in both uniformity and spread performance on all the real networks. It can be observed that the obtained Pareto optimal solutions are sufficient and uniformly and widely spread, so the decision-makers can choose proper feasible solutions from the obtained Pareto optimal solution set according to the practical needs.

5.2.6. Running time of the proposed algorithm

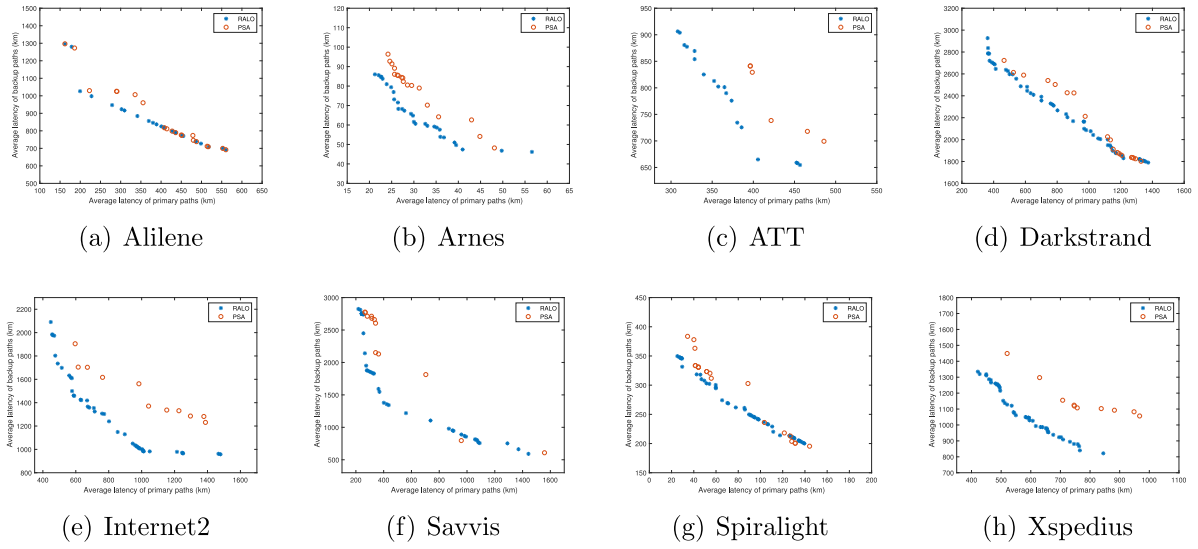
We compare the running time of algorithm RALO with algorithm PSA since only these two algorithms are multi-objective optimization algorithms and algorithm PSA performs better than algorithms LARC and Survivor. Algorithm RALO and PSA are implemented in C++ and run on a platform with Intel(R) Core(TM) 5-9300H 2.40 GHz CPU, 8 GB RAM, and 64-bit version of Windows 10. We execute algorithms RALO and PSA on all the real network topologies with 10 controllers. The average running time of the algorithms on each network topology is shown in Table 8.

The average running time of algorithms RALO and PSA on all the network topologies is 6.559 s and 2.35 s respectively. The running time of algorithm RALO is more than that of algorithm PSA since algorithm RALO searches a wider solution space than PSA. However, it can be seen that in all the network topologies, the running time of both algorithms RALO and PSA is within 20 s. In practice, the running time of algorithm RALO is acceptable.

5.2.7. Impact of the important components of the proposed algorithm

We assess the impact of important components of the proposed algorithm RALO. We disable some components at a time and run the algorithm. We investigate four cases: (a) RALO with *perturbation* missing; (b) RALO with *shake* disabled; (c) RALO without *perturbation* and *shake*; (d) RALO with all the components available.

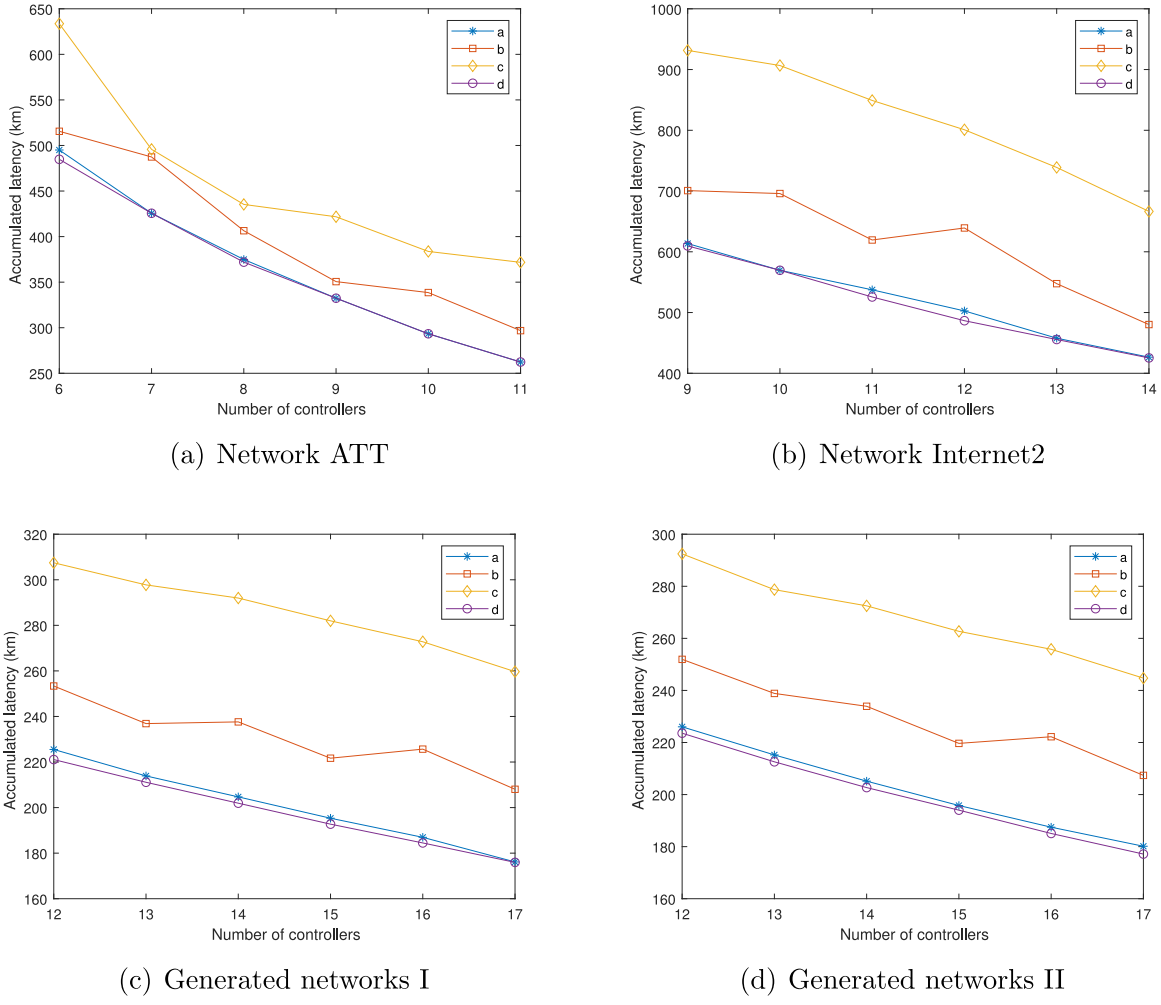
Figs. 6(a)–(d) depict the average accumulated latency with different number of controllers for algorithm RALO on ATT, Internet2, generated networks I and generated networks II, respectively. It can be observed that algorithm RALO results in the worst performance among all the cases when both the operations of *shake* and *perturbation* are missing in the algorithm. The results of this case are worse than those obtained by the algorithm when all the components are available 16.5%–62.2% on all the four kinds of networks. Therefore, the operations of *shake* and *perturbation* are important for the algorithm to jump out of the local optimal solution. On the four kinds of networks, the results achieved by the case without operation *perturbation* are better than those obtained by the algorithm without operation *shake* 2.8%–27.2%, which shows that operation *shake* plays a more important role in avoiding falling in local optimum than operation *perturbation*. The algorithm achieves the best performance among all the cases when all the components are available, while the case without component *perturbation* obtains better performance than the other two cases. The case with all the components available outperforms the case without component *perturbation* by up to 2.1%, 3.3%, 2.0%, and 1.6% for the networks of ATT,



(a) Alilene (b) Arnes (c) ATT (d) Darkstrand

(e) Internet2 (f) Savvis (g) Spiralight (h) Xspedius

Fig. 5. Pareto frontier corresponding to real network topologies.



(a) Network ATT (b) Network Internet2

(c) Generated networks I (d) Generated networks II

Fig. 6. The average accumulated latency with different components available in the algorithm.

Table 7
SP and MS values of the Pareto optimal solution sets.

Metric	Method	Alilene	Arnes	ATT	Darkstrand	Internet2	Savvis	Spiralight	Xspedius
SP	RALO	18.1	1.7	11.7	19.1	26.4	74.3	3.0	14.7
	PSA	22.8	3.5	25.7	66.2	70.6	322.8	12.8	88.5
MS	RALO	724.0	53.2	292.3	1521.7	1531.7	2547.3	188.3	665.0
	PSA	724.0	50.4	167.7	1262.2	1039.3	2524.4	172.2	594.9

Table 8
The average running time (s) of the algorithms on different network topologies.

Algorithm	Network									
	Abilene	Arnes	ATT	Darkstrand	Internet2	Savvis	Spiralight	Xspedius	ER	SW
RALO	0.29	6.94	2.41	6.41	9.58	1.01	3.07	7.92	16.59	11.37
PSA	1.01	2.38	1.52	1.91	2.49	1.13	1.05	2.46	4.84	4.71

network Internet2, generated networks I, and generated networks II, respectively.

6. Conclusions

In this paper, we formulated a novel multi-objective SDN controller placement problem to minimize the switch-to-controller communication delay for both the cases without link failure and with single-link-failure. We proposed an efficient metaheuristic-based Reliability-Aware and Latency-Oriented controller placement algorithm (RALO) for multi-objective multiple controller placement. The algorithm constructs an initial feasible solution by a greedy method with network partition, then repeatedly generates new solutions with variable neighborhood search. Once a new solution is generated, the algorithm decides whether to accept the new solution as a non-dominated solution to the problem and performs update operation on the Pareto optimal solution set. To avoid falling into the local optimum, the algorithm also performs perturbation and destruction operations on the current solution. We conducted simulations on eight real networks from Internet topology zoo and two kinds of generated networks conforming to ER (Erdos–Renyi) random model and small-world model. The simulations results demonstrated that the proposed algorithm could achieve a competitive performance of switch-to-controller latencies in both the cases without link failure and with single-link failure, and the accumulated delay of primary and backup paths between the controllers and the switches. The widely-spread Pareto optimal solution set provided by algorithm RALO allows network administrators with flexible choices to achieve a balance between the switch-to-controller delay of primary and backup paths.

CRediT authorship contribution statement

Yuqi Fan: Conceptualization, Methodology, Writing. **Lunfei Wang:** Methodology, Simulations, Validation. **Xiaohui Yuan:** Validation, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was partly supported by the National Natural Science Foundation of China (U1836102).

References

[1] B.A.A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, T. Turletti, A survey of software-defined networking: Past, present, and future of programmable networks, IEEE Commun. Surv. Tutor. 16 (3) (2014) 1617–1634.
[2] D. Kreutz, F.M.V. Ramos, P.E. Veríssimo, C.E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-defined networking: A comprehensive survey, Proc. IEEE 103 (1) (2015) 14–76.
[3] S.H. Yeganeh, Y. Ganjali, Kandoo: a framework for efficient and scalable offloading of control applications, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN'12, Helsinki, Finland, 2012, pp. 19–24.
[4] A. Tootoonchian, Y. Ganjali, HyperFlow: A distributed control plane for openflow, in: 2010 Internet Network Management Conference on Research on Enterprise Networking, INM/WREN'10, San Jose, CA, 2010.
[5] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, S. Shenker, Onix: A distributed control platform for large-scale production networks, in: Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, Berkeley, CA, 2010, pp. 351–364.
[6] B. Heller, R. Sherwood, N. McKeown, The controller placement problem, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN'12, Helsinki, Finland, 2012, pp. 7–12.
[7] Y. Zhang, L. Cui, W. Wang, Y. Zhang, A survey on software defined networking with multiple controllers, J. Netw. Comput. Appl. 103 (2018) 101–118.
[8] Y. Zhang, N. Beheshti, M. Tatipamula, On resilience of split-architecture networks, in: 2011 IEEE Global Telecommunications Conference, GLOBECOM, Kathmandu, Nepal, 2011, pp. 1–6.
[9] G. Yao, J. Bi, Y. Li, L. Guo, On the capacitated controller placement problem in software defined networks, IEEE Commun. Lett. 18 (8) (2014) 1339–1342.
[10] Y. Fan, T. Ouyang, Reliability-aware controller placements in software defined networks, in: The 21st IEEE International Conference on High Performance Computing and Communications, HPCC 2019, Zhangjiajie, China, 2019.
[11] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, Y. Ganjali, C. Diot, Characterization of failures in an IP backbone, in: 2004 IEEE International Conference on Computer Communications, INFOCOM, Vol. 4, Hong Kong, China, 2004, pp. 2307–2317.
[12] M.F. Bari, A.R. Roy, S.R. Chowdhury, Q. Zhang, M.F. Zhani, R. Ahmed, R. Boutaba, Dynamic controller provisioning in software defined networks, in: Proceedings of the 9th International Conference on Network and Service Management, CNSM 2013, Zurich, Switzerland, 2013, pp. 18–25.
[13] 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: 2014 IEEE 15th International Symposium on World of Wireless, Mobile and Multimedia Networks, WoWMoM, Sydney, NSW, Australia, 2014, pp. 1–6.
[14] M. Tanha, D. Sajjadi, R. Ruby, J. Pan, Capacity-aware and delay-guaranteed resilient controller placement for software-defined WANs, IEEE Trans. Netw. Serv. Manag. 15 (3) (2018) 991–1005.
[15] L. Yao, P. Hong, W. Zhang, J. Li, D. Ni, Controller placement and flow based dynamic management problem towards SDN, in: 2015 IEEE International Conference on Communication Workshop, ICCW, London, UK, 2015, pp. 363–368.
[16] A. Sallahi, M. St-Hilaire, Optimal model for the controller placement problem in software defined networks, IEEE Commun. Lett. 19 (1) (2015) 30–33.
[17] M.T.I. ul Huque, G. Jourjon, V. Gramoli, Revisiting the controller placement problem, in: 2015 IEEE 40th Conference on Local Computer Networks, LCN, Clearwater Beach, FL, 2015, pp. 450–453.
[18] 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, ICC, Kuala Lumpur, Malaysia, 2016, pp. 1–6.

[19] M. Jia, W. Liang, M. Huang, Z. Xu, Y. Ma, Routing cost minimization and throughput maximization of NFV-enabled unicasting in software-defined networks, IEEE Trans. Netw. Serv. Manag. 15 (2) (2018) 732–745.
[20] Y. Hu, W. Wendong, X. Gong, X. Que, C. Shiduan, Reliability-aware controller placement for software-defined networks, in: 2013 IFIP/IEEE International Symposium on Integrated Network Management, IM 2013, Ghent, Belgium, 2013, pp. 672–675.
[21] Y. Hu, W. Wang, X. Gong, X. Que, S. Cheng, On reliability-optimized controller placement for software-defined networks, China Commun. 11 (2) (2014) 38–54.
[22] M. Guo, P. Bhattacharya, Controller placement for improving resilience of software-defined networks, in: 2013 Fourth International Conference on Networking and Distributed Computing, Los Angeles, CA, 2013, pp. 23–27.
[23] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, P. Tran-Gia, Pareto-optimal resilient controller placement in SDN-based core networks, in: Proceedings of the 2013 25th International Teletraffic Congress, ITC, Shanghai, China, 2013, pp. 1–9.
[24] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, M. Hoffmann, Heuristic approaches to the controller placement problem in large scale SDN networks, IEEE Trans. Netw. Serv. Manag. 12 (1) (2015) 4–17.
[25] L. Müller, R.R. Oliveira, M.C. Luizelli, L.P. Gaspary, M.P. Barcellos, Survivor: an enhanced controller placement strategy for improving SDN survivability, in: 2014 IEEE Global Communications Conference, GLOBECOM, Austin, TX, 2014, pp. 1909–1915.
[26] D. Santos, A. de Sousa, C.M. Machuca, Robust SDN controller placement to malicious node attacks, in: 2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops, ICIN, IEEE, IEEE Commun Soc, DNAC, ACM, Sigmobile, Gandi, Nokia, Orange, 2018, 21st International Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), Paris, FRANCE, FEB 19-22, 2018.
[27] P. Vizarreta, C.M. Machuca, W. Kellerer, Controller placement strategies for a resilient SDN control plane, in: 2016 8th International Workshop on Resilient Networks Design and Modeling, RNDM, Halmstad, Sweden, 2016, pp. 253–259.
[28] F.J. Ros, P.M. Ruiz, On reliable controller placements in software-defined networks, Comput. Commun. 77 (2016) 41–51.
[29] H. Huang, S. Guo, W. Liang, K. Li, B. Ye, W. Zhuang, Near-optimal routing protection for in-band software-defined heterogeneous networks, IEEE J. Sel. Areas Commun. 34 (11) (2016) 2918–2934.
[30] Y. Fan, Y. Xia, W. Liang, X. Zhang, Latency-aware reliable controller placements in SDNs, in: 11th EAI International Conference on Communications and Networking in China, CHINACOM 2016, Chongqing, China, 2016.
[31] IBM, CPLEX optimizer, URL http://https://www.ibm.com/analytics/cplex-optimizer, (Accessed on: 28 Feb 2020).
[32] The internet topology zoo, http://www.topology-zoo.org.
[33] Stanford network analysis project, http://snap.stanford.edu.
[34] J.R. Schott, Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization, Tech. Rep., Air force inst of tech Wright-Patterson afb OH, 1995.
[35] E. Zitzler, K. Deb, L. Thiele, Comparison of multiobjective evolutionary algorithms: empirical results, Evol. Comput. 8 (2) (2000) 173–195.