Computer Networks 112 (2017) 24–35

Contents lists available at ScienceDirect

Computer Networks

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

Density cluster based approach for controller placement problem in large-scale software defined networkings

Jianxin Liao[a,*], Haifeng Sun[a], Jingyu Wang[a], Qi Qi[a], Kai Li[a], Tonghong Li[b]

[a] *State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, 100876, China*
[b] *Department of Computer Science, Technical University of Madrid, Madrid, 28660, Spain*

ARTICLE INFO

Article history:
Received 14 June 2016
Revised 23 September 2016
Accepted 28 October 2016
Available online 29 October 2016

Keywords:
SDN
Controller placement
Density based clustering
Resilience
Failure tolerance
Latency

ABSTRACT

Software Defined Networking (SDN) decouples control and data planes. The separation arises a problem known as the controller placement, i.e., how many and where controllers should be deployed. Currently, most works defined this problem as the multi-objective combinatorial optimization problem and used heuristic algorithms to search the optimal solution. However, these heuristic algorithms have the drawback of being easily trapped in local optimal solutions and consuming high time. In this paper, we propose an approach named as Density Based Controller Placement (DBCP), which uses a density-based switch clustering algorithm to split the network into several sub-networks. As switches are tightly connected within the same sub-network and less connected from the switches in other sub-networks, we deploy one controller in each sub-network. In DBCP, the size of each sub-network can be decided by the capacity of the controller deployed. Moreover, the optimal number of controllers is obtained according to the density-based clustering. We evaluate DBCP's performance on a set of 262 publicly available network topologies. The experimental results show that DBCP provides better performance than the state-of-the-art approaches in terms of time consumption, propagation latency, and fault tolerance.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

With the introduction of Software Defined Networking (SDN), the separation of control plane and data plane simplifies the networking management and improves its scalability [1]. The controller in control plane manages switches by providing them with rules that dictate their packet handling behavior. In a large-scale network, a good placement best utilizes existing network connectivity among the switches [2]. A fast response and reliable connection between the switch and the associated controller is a key point for SDN networks [3]. A single controller is hard to control all the switches in a large-scale SDN network, because the capability of the controller is limited and the propagation latency between the controller and switches is very large [3]. Currently, most researches aim to deploy multiple controllers at different locations to corporately control the whole data plane [2,4–6]. In this kind of architectures, the placement of multiple controllers becomes a critical problem.

As discussed in some related work, the controller placement is a complexity optimization problem [7,8], where the following factors should be taken into consideration whenever designing a placement strategy:

(1) **The latency of control signaling.** The switches receive the instructions on how to forward the new flows. Whenever the latency between controllers and switches reaches a threshold, the latency on the whole network will increase substantially. In this case, the controller processing latency is a non-negligible factor in the total round-trip latency [4].
(2) **The server capacity limitation.** Due to the constraints of the resources such as processors, memory, and access bandwidth, a commodity server only has the capacity to manage a limited number of switches. On the other hand, the overload of controllers may decrease the performance of SDN [5].
(3) **The required number of controllers.** In large-scale SDN networks, a large amount of switches in data plane construct a complex networks. It is difficult for administrators to figure out how many controllers should be deployed. Some work uses the traversal searching method to iteratively find the best performance number, which may lead to high time consumption [6].
(4) **Fault tolerance.** Unlike the traditional network architecture, the switches do not have control ability due to the split architecture of SDN. Each switch is assigned a controller. Therefore,

* Corresponding author.
E-mail addresses: jxlbupt@gmail.com (J. Liao), hfsun@bupt.edu.cn (H. Sun), wangjingyu@bupt.edu.cn (J. Wang), qiqi8266@bupt.edu.cn (Q. Qi), likai@ebupt.com (K. Li), tonghong@fi.upm.es (T. Li).

http://dx.doi.org/10.1016/j.comnet.2016.10.014
1389-1286/© 2016 Elsevier B.V. All rights reserved.

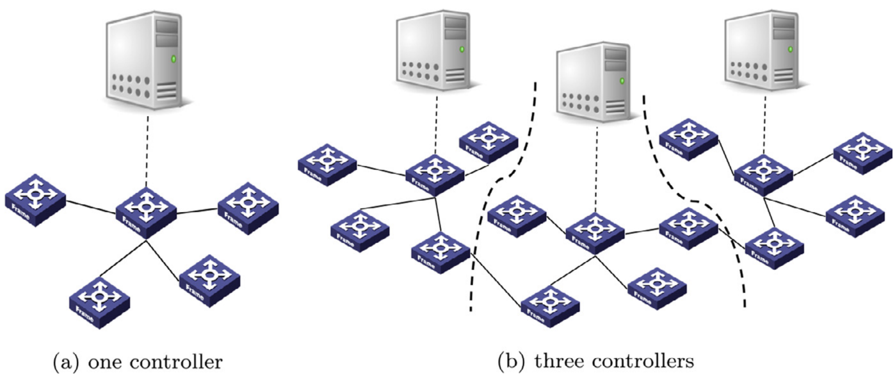


Fig. 1. Two examples of controller placement:(a) a star topology data plane and one controller. (b) a three star topologies constructed data plane and three controllers. The dotted curves represent the switches clustering result.

whenever a switch loses the connection to its controller, it will no longer receive any new routing instructions and thus drop all packets [2].

(5) **Inter-controller communication.** In multi-controller SDNs, each switch is controlled by a specific controller. If a controller wants to send messages to a switch controlled by another controller, the controllers need to communicate with each other [9]. Therefore, the inter-controller communication affects the performance in end-to-end communication between disparate switches controlled by different controllers.

To our best knowledge, there is no strategy to take into account all these factors for solving the controller placement problem. A well-known controller placement strategy, which is introduced in [3], motivated by minimizing the propagation latency between switches and controllers. Without considering the capacity limitation of the controller, this strategy is not always applicable. A good placement should minimize the propagation latency, whereas the load of each controller should not exceed its capacity. Yao et al. [5] defined a Capacitated Controller Placement Problem (CCPP) to consider the controller's capacity while minimizing the average propagation latency. Recently, some work, such as Pareto-based Optimal COntroller placement (POCO) [7] and Min-cut strategy [2], considered the reliability analysis of the networks. However, these methods cannot be applied to the CCPP problem.

Actually, the structure of the data plan is an important clue to find the optimal placement. As shown in Fig. 1a, a controller is deployed at the center switch in a star topology network, which is the optimal placement. In Fig. 1b, a data plane is constructed by three star topologies. In this network, the placement with three controllers is optimal in terms of latency and reliability.

For this purpose, we propose a new placement approach named as Density Based Controller Placement (DBCP) to solve the above problems. In this approach, we maintain a table of all the switch densities and the relevant information, which are newly defined in this article and will be discussed in detail at Section 3. Based on this table, the network is split into several sub-networks by suing a density-based clustering method [10] according to the network architecture. Then, the best placements of controllers can be selected by traveling all the candidate locations in each sub-network, and the SDN network is constructed by connecting all switches to their nearest controllers. Our proposed algorithm is a fast response and stable solution, which can be easily applied in real networks. Our critical contributions are as follows:

1) We use a fast density-based clustering method to cluster the data plane, where an optimal required number of controllers can be given. This method is faster for clustering, compared with the conventional iteration based clustering methods, such as k-means.
2) We propose a new strategy, which significantly improves the performance of control plane. Our experiment results show our approach is better than the state-of-the-art approaches.

The remainder of this paper is organized as follows: Section 2 surveys the related work. Section 3 defines the problem and proposes the design of DBCP. Section 4 presents some analysis regarding the recommended controller numbers and parameter. Section 5 provides the experiment results. Finally, discussions and conclusions are presented in Section 6.

2. Related work

Since Onix [11], Hyperflow [12], and Devoflow [13] were used as the distributed control architecture to solve the problem of scalability and reliability of SDN, more and more researchers have studied the controller placement problem. The controller placement problem (CPP) was first defined in [3]. The CPP problem mainly concerns on two questions: how many and where controllers should be deployed. It was proved as a NP-hard problem [3].

Heller et. al. [3] studied the best controller placement solutions that minimize the controller-to-switch propagation latency, which includes the average latency and the worst-case latency (or maximum latency). Sallahi et. al. [4] considered the cost of controllers, such as the cost of installing controllers, lining the controllers, and linking the controllers together. Both in [3] and [4], they all used a traversal method to search all the candidate solutions to find the optimal one. Traversal-based methods can provide the best performance solutions. However, the time consumption is extremely high in a large-scale network. As described in their papers, there are a lot of topologies that cannot be solved within 30 hours. Besides, Yao et. al. [5] considered that the load of the controller should not exceed its capacity, and defined a capacitated controller placement problem (CCPP). To solve the CCPP problem, they proposed an advanced capacity K-center algorithm [14] to search the best placement solutions, which travels different k values to find a least k to meet the capacity requirements. In [15], Yao et. al. found that the controller placement problem is a pre-planning problem, where the flow is varied. With the varied flow, some of the controllers may be overloaded. They proposed a method to place the controllers at hotspot where the switches carry the most flow. The switches with low flow can dynamically migrate from an overloaded controller to the other controller.

Some other works of CPP considered the reliability of networks. In [16], the authors defined a fault tolerant controller placement problem (FTCP) and found that the required controller number is positively correlated with the number of spokes (nodes with degree one) in a network. Based on these, they adopted a heuristic algorithm to compute the placements to meet the reliability and gave an upper bound on the number of controllers. For the same problem, Zhang et. al. [2] used a Min-cut method [17] to compute the maximal disjoint paths to separate the network into several sub-networks. Given an initial bisection of the network, they try to find a sequence of switch pair exchanges that leads to a reduction of the cut size. The cut size is calculated as the number of links need to be cut to split the network. This process runs recursively until no further reduction can be obtained by changing any pairs. Then, the controllers are deployed to the centroid in each sub-network respectively.

In [18], Guo et. al. firstly divided a generated hierarchical tree [19] of nodes into k clusters, each of which is a sub-network managed by a controller. Then, they selected a node with the maximal closeness to all the other nodes in the sub-network to deploy the controllers. In [20], they defined two problems, the controller placement under comprehensive network state problem (CPCNS) and the controller placement under single link failure problem (CP-SLF), to evaluate the reliability of networks, where a traversal algorithm and a greedy-based algorithm are used to find the best solution to meet the requirement of these two problems. In addition, Hu et. al. [21] compared different methods to solve the reliability problem, such as l-w-greedy, simulated annealing [22], and brute force. They found that the brute force is the best solution without considering the time consumption.

Some works solved the CPP with more than one metrics, e.g. latency and reliability. In [7,8], they used the heuristics based algorithm [23] to search all the candidate solutions of Pareto-optimal placements with respect to different performance metrics. Lange et. al. [7] proposed a framework named as Pareto-based Optimal COntroller placement (POCO). It explores a subset of the search space by Pareto simulated annealing (PSA) and return the Pareto frontier of this subset. The PSA explores search space by moving to the neighbor placement with a slow decrease in the probability of adding worse placement to the search space. The multiple objectives and case specific requirements can be added into the evaluation process. The benefit of multiple objectives is that the more appropriate choices could be obtained by varying the trade-off.

Most of these works are based on a given number of controllers. However, it is impossible to know, in advance, how many controllers are required in a large-scale network. The only way to find this number is to compare the results by traversing all the candidate numbers, which is infeasible in a large-scale network.

3. Solution

The data plane network topology $G(S, L)$ contains a set of switches S and a set of bi-directional links L. In this paper, we define a link l_{ij} between switch s_i and s_j as:

$$l_{ij} = \begin{cases} 1, & \text{if there is a link between } s_i \text{ and } s_j; \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Different from other methods, we analyze the topology structures and split the networks into several sub-networks. Obviously, the switches should have close connections within the same sub-network, while they should have relatively few connections to the switches in other sub-networks. The sub-network should be the least separable network. After splitting, the sub-network has high fault tolerant performance. At the same time, the sub-network latency performance cannot be increased dramatically by finer grained separation. In summary, we can benefit from the network splitting in the following several points:

First, we can obtain the optimal number of controllers to be placed in an arbitrary network.

Second, we can obtain the relatively stable sub-networks. This means we can decrease the probability of deploying a controller at high-risk locations.

Last, the multiple-controller placement problem can be simplified as the one-controller placement problem, which is easy to solve.

3.1. Split the network

In DBCP, we split the network by clustering the switches. For each switch s_i, we compute two quantities: its local density ρ_i and its distance δ_i to switches with higher density. These quantities only depend on the distances between switches. The local density ρ_i of switch s_i is defined as:

$$\rho_i = \sum_j \chi(d_{ij} - d_c) \tag{2}$$

where the d_{ij} represents the distance between switches d_i and d_j. The d_c is a threshold distance. Only the distance between switches lower than d_c will be considered as a closed switch. And the function χ is defined as:

$$\chi(x) = \begin{cases} 1, & \text{if } x < 0; \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Basically, ρ_i is equal to the number of switches whose distances to switch s_i are less than d_c. For large-scale networks, d_c is defined as 0.3 times the graph diameter, which is discussed in detail at Section 4.2.

δ_i is measured by computing the minimum distance between switch s_i and any other switch with higher density:

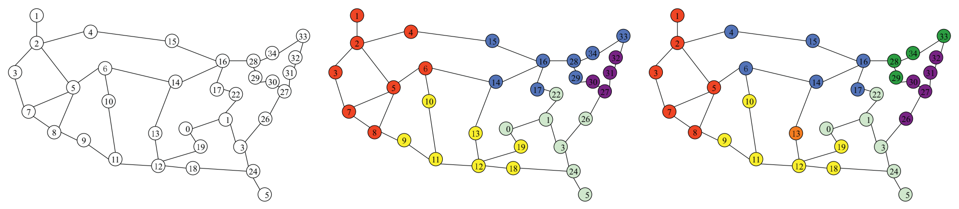
$$\delta_i = \min_{j:\rho_j > \rho_i} (d_{ij}) \tag{4}$$

For the switch with the highest density, we conventionally take $\delta_i = \max_j(d_{ij})$. Thus, the cluster centers are the switches whose δ are anomalously large. Algorithm 1 shows the details of finding the optimal number of controllers, where k represents the optimal recommended controller number.

Algorithm 1 Analyzing process.

```
1.  input: $G = (S, L)$, $d_c$
2.  $k = 0$
3.  for $s$ in $S$:
4.    $\rho_s$=GetNumberOfNodesWithinDistance($s$, $d_c$, $G$)
5.  end for
6.  for $s$ in $S$:
7.    $\delta_s$=MinDistanceToHigherDensityNode($s$, $\rho$, $G$)
8.    $\bar{\delta} = \frac{1}{|S|} \sum_{s \in S} \delta_s$
9.    if $\delta_s > \bar{\delta}$:
10.     $k+ = 1$
11.   end if
12. end for
13. return $k$
```

Here, we can set any number of clusters. If the number is smaller than k, the switches with higher ρ and δ will have the priority of being set as the cluster centers. If the number is greater



(a) The original switch networks tructure

(b) Switch clustering result for CPP problem

(c) Switch clustering result for CCPP problem

Fig. 2. The clustering results performed by proposed method for different scenarios in the Internet2 OS3E. (The colors represent the clusters.). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

than k, all the switches with $\delta > \bar{\delta}$ will be set as the cluster centers and the other switches with higher ρ will have the priority of becoming the cluster centers. After the cluster centers are obtained, each remaining point is assigned to the same cluster as its nearest neighbor with higher density. Here, we assume that the capacity of controllers is not considered, and the cluster assignment can be performed directly. Algorithm 2 shows the details of clustering process.

Algorithm 2 Clustering process.

```
1. input: $G = (S, L)$, $\rho$, $\delta$
2. input: the number of controller $k$
3. max$_k$ $S$=FindTheNodesAsClusterCenter($k$, $\rho$, $\delta$, $G$)
4. for $s$ in $S$:
5.   if $s \in$ max$_k$ $S$:
6.     $s$ belongs to a new cluster
7.   else:
8.     $s$ belongs to the cluster of nearest higher density node
9.   end if
10. end for
```

We use a set of switches shown in Fig. 2a as an example, which is named as Internet2 OS2E network[1] with 34 nodes and 41 edges. Each node represents a switch and each line between switches represents the link in Fig. 2a. We assume that the bandwidths are the same, so the distances between switches are defined as the number of the shortest path hops. We set $d_c = 2$ here, which means we only consider the switches within 2-hop radius. The ρ and δ values of switches are calculated by (2) and (4) respectively. The results are shown in Table 1 and Fig. 3. We can see that the switches 5, 12, 16, and 23 have the highest δ and a relatively high ρ. So we identify them as the cluster centers. The value of δ and ρ for switch 27 are higher than the average value, so switch 27 can also be identified as a cluster center. We can cluster the network into 5 clusters, which means 5 controllers are needed. Some switches, such as switches 11 and 23, have similar ρ but different δ. The switches with lower value of δ are assigned to the same cluster as the close switch with higher ρ value. The higher value of δ indicates that these switches are far away from the cluster center, thus they should be assigned a new cluster center. After the cluster centers are obtained, the other switches are assigned to the same cluster as its nearest neighbor with higher density. The result clusters are shown in Fig. 2b. In this figure, we can see that the network is split into 5 parts, and the switches with the same color represent a cluster.

Table 1
The ρ and δ values of switches in Fig. 2a.

Switch number	ρ	δ	Switch number	ρ	δ
1	4	1	18	7	1
2	8	1	19	6	1
3	6	1	20	5	1
4	6	1	21	7	1
5	9	3	22	5	1
6	8	1	23	8	3
7	6	1	24	6	1
8	7	1	25	3	1
9	6	1	26	6	1
10	6	1	27	6	2
11	8	1	28	8	1
12	9	3	29	5	1
13	7	1	30	5	1
14	9	1	31	5	1
15	6	1	32	4	1
16	10	3	33	4	1
17	6	1	34	8	1

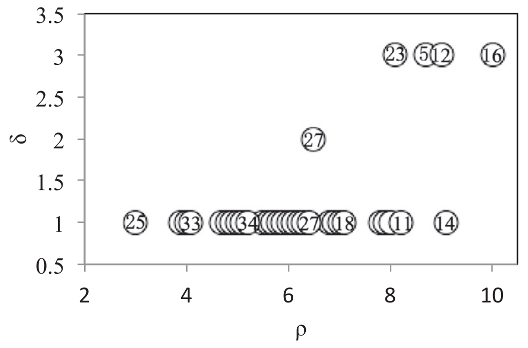


Fig. 3. Observation graph for the data in Table. 1. (The values of ρ are integers, we stagger them in order to observation.)

3.2. Split the network considering capacity

In physical networks, the load of controllers also influences the network performance. A Capacitated-DBCP (CDBCP) can be used to solve the CCPP problem.

Considering the capacity of controllers, the main problem is the clusters can't be as large as it should be. Assuming each controller θ has a capacity $L(\theta)$. The load of controlling switch s is denoted by $l(s)$. $L(\theta)$ and $l(s)$ could be considered as the number of controller lookups, proactive and reactive flow installation schemes, or the combination of them. Then the constraint condition is defined as

[1] Internet2 open science, scholarship and services exchange. http://www.internet2.edu/network/ose/.

follows:

$$L(\theta) \geq \sum_{s \in \mathbb{S}(\theta)} l(s), \forall \theta \in \Theta \tag{5}$$

where $\mathbb{S}(\theta)$ represents the collection of switches in the sub-network which are controlled by controller θ, and Θ represents all the controllers.

If a cluster contain enough load of controlling and cannot expand any switches according to the constraints, the other switches consider the other adjacent clusters or become a new cluster. In order to keep the attribute of the switches in same cluster are closely connected, we priority considering to assigned the closely connected switches into a same cluster, and the switches which are at cluster border as candidate switches to fill the clusters which can expand the size.

To find the switches at cluster border, we compute another quantity for each switch: its borderline indexing σ, which represents the uncertainty of a switch belonging to different clusters. For each switch, it can be assigned to the linked switch with equal or higher density . If the densities of equal or higher density neighbors are similar, the σ of the switch is higher, which means it is a border switch. We use the information entropy theory to calculate σ_i for switch s_i as follows:

$$td_i = \sum_{j}^{s_j \in UL(s_i)} \rho_j \tag{6}$$

$$\sigma_i = \sum_{j}^{s_j \in UL(s_i)} \frac{\rho_j}{td_i} \log_{|UL(s_i)|} \frac{\rho_j}{td_i} \tag{7}$$

where the $UL(s_i)$ represents the set of higher and equal density switches linked to s_i.

In our clustering assignment, the switches with lower value of σ have priority. The switch is assigned to the same cluster as its nearest neighbor with higher density, unless the load of switches in this cluster exceeds the capacity of the controller. If there is no cluster to be assigned, the switch becomes a new cluster with itself. Algorithm 3 shows the details of clustering the network considering the capacity.

Assuming the load of controlling a switch is the same and each controller can control 6 switches maximum. In Fig. 2c, we show an example of switch clustering for the CCPP problem. 34 switches are split into 7 different clusters, where switches of different clusters are labeled with different colors. Switch 13 is a single switch in its cluster, because its neighbor clusters are full and it can not be assigned to any of them. (Table 2)

Algorithm 3 Clustering process for CCPP problem.

```
1. input: $G = (S, L)$, $\rho$, $\delta$
2. input: the capacity of controllers $L(\theta)$
3. for $s$ in $S$:
4.   $\sigma_s$=GetTheBorderlineBasedOnNeighbors(s,G)
5. end for
6. sort $S$ according to $\sigma$ in ascending order
7. Each switch $s$ forms a cluster $N(s)$
8. for $s$ in $S$:
9.   Get the neighbor switches with higher or equal $\rho$ of $s$ as
     $UL(s)$
10.  sort $UL(s)$ according to $\rho$ in descending order
11.  for $s'$ in $UL(s)$
12.    Get the current cluster $N(s)$ and $N(s')$ that contain $s$ and
       $s'$ respectively
13.    if $L(\theta) \geq \sum_{s_i \in (N(s)+N(s'))} l(s_i)$:
14.      $s$ belongs to the cluster $N(s')$
15.      break
16.    end if
17.  end for
18. end for
```

Table 2
The ρ, δ, and σ values of switches in Fig. 2a.

Switch number	ρ	δ	σ	Switch number	ρ	δ	σ
1	4	1	0.0	18	7	1	0.0
2	8	1	0.0	19	6	1	0.0
3	6	1	0.98	20	5	1	0.99
4	6	1	0.98	21	7	1	0.0
5	9	3	0.0	22	5	1	0.99
6	8	1	1.0	23	8	3	0.0
7	6	1	0.98	24	6	1	0.99
8	7	1	0.0	25	3	1	0.0
9	6	1	0.99	26	6	1	0.98
10	6	1	1.0	27	6	2	0.0
11	8	1	0.0	28	8	1	0.99
12	9	3	0.0	29	5	1	0.96
13	7	1	1.0	30	5	1	0.99
14	9	1	0.0	31	5	1	0.0
15	6	1	0.95	32	4	1	0.99
16	10	3	0.0	33	4	1	0.91
17	6	1	0.0	34	8	1	0.0

3.3. Controller placement

After splitting the network, we should place the controllers for all sub-networks. Note that in each sub-network, only a single controller is deployed. We use the combination of optimal solutions for each sub-network to approximate the optimal solution for the whole network. The placement of controllers can be decided according to different objective functions. More generally, the latency and reliability are used.

3.3.1. Controller-to-switch latency

For a sub-network graph, the average propagation latency for the placement location v of controller θ is calculated as:

$$\pi^{avglatency}(\mathbb{S}(\theta)) = \min_{v \in \mathbb{S}(\theta)} \frac{1}{|\mathbb{S}(\theta)|} \sum_{s \in \mathbb{S}(\theta)} d(v, s) \tag{8}$$

where $d(v, s)$ is the latency from the controller location v to switch s, which is defined as the number of hops between them in this paper. $\mathbb{S}(\theta)$ represents the collection of switches should be controlled by controller θ.

The objective function of the worst-case latency with respect to controller θ is calculated as:

$$\pi^{maxlatency}(\mathbb{S}(\theta)) = \min_{v \in \mathbb{S}(\theta)} \max_{s \in \mathbb{S}(\theta)} d(v, s) \tag{9}$$

3.3.2. Inter-controller latency

To reduce the communication cost among controllers, the controllers should be deployed as closely as possible. Intuitively, we need to minimize the latency among them. However, they are deployed independently. The latency of inter-controller communication cannot be directly measured by computing the latency among them.

To minimize the latency from a controller to other controllers, we should deploy that controller as closely as to other sub-networks. The latency from a controller to the switch of other sub-networks can be measured. So, the minimum latency from a controller to other controllers could be measured by calculating the latency from its placement to all the other sub-network's switches. The objective function of the inter-controller latency with respect to controller θ is calculated as:

$$\pi^{inter-controller}(\mathbb{S}(\theta)) = \min_{v \in \mathbb{S}(\theta)} \frac{1}{|S|} \sum_{s \in (S-\mathbb{S}(\theta))} d(v, s) \tag{10}$$

where S represents all the switches in the network.

3.3.3. Network reliability

To evaluate the network's reliability, we use a failure objective function to calculate the average and the worst-case number of switches that lose their connections to the controller when one or two links faults occur. A link fault means that the traffic traversing the link can no longer be transferred over the link. We assume that all the links fail independently. The objective function of the average controller-loss switch $\pi^{avgloss}$ is defined as:

$$\pi^{avgloss}(\mathbb{S}(\theta)) = \min_{v \in \mathbb{S}(\theta)} \frac{1}{|P|} \sum_{p \in P} \frac{1}{|S|} \sum_{s \in \mathbb{S}(\theta)} e^{p}_{v,s} \tag{11}$$

where P represents all the link faults. $e^{p}_{v,s}$ represents whether or not switch s loses its connection[2] to controller location v and it is defined as:

$$e^{p}_{v,s} = \begin{cases} 1, & \text{if network fault at } p,\ s \text{ loss connection to } v; \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

The objective function of the worst-case controller-loss switches $\pi^{maxloss}$ is defined as:

$$\pi^{maxloss}(\mathbb{S}(\theta)) = \min_{v \in \mathbb{S}(\theta)} \max_{p \in P} \sum_{s \in \mathbb{S}(\theta)} e^{p}_{v,s} \tag{13}$$

One or multiple, possible conflict, objective functions can be used to evaluate each switch in a sub-network. We can either find a best placement or a set of Pareto-optimal placements by using a traversal method within each sub-network.

4. Analysis on DBCP

In this section, we analyze the performance of DBCP by using hundreds of topologies in the Internet Topology Zoo, which are a collection of annotated network graphs derived from the public network maps [24]. We employ this data set because it covers a diverse range of geographic areas (regional, continental, and global), network sizes (8–200 nodes), and topologies (line, ring, hub-and-spoke, tree, and mesh). The graphs in the Zoo do not conform to any single model. All the isolated nodes are removed in these topologies.

4.1. Evaluation of k

Since the controller placement was first talked in [3], the question of “how many controllers are needed” is not well solved. Some works solved this problem by using the least number of controllers to meet the given limitations, such as the controller capacity and the reliability standard. As the controller placement is a

[2] Loss the connection is defined as there is no available path between the switch and the controller in whole network.

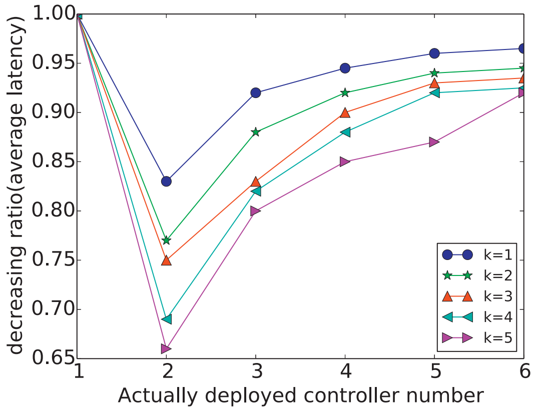


Fig. 4. Average latency decreasing ratios of different category topologies at different numbers of controller.

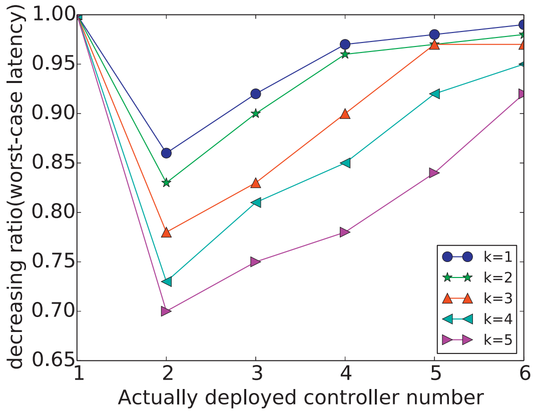


Fig. 5. Worst-case latency decreasing ratios of different category topologies at different numbers of controller.

Multi-objective combinatorial optimization (MOCO) problem [7]. It is hard to obtain the optimal number of controllers to be placed in a network. Generally, if we place more controllers in a network, the performance will be better. However, an efficient number of controllers, i.e., the least number of controllers with relatively best performance, is very important for the controller placement problem [3]. To find the point of diminishing return, DBCP divides the switches into different sub-networks according to the network structure. In the following, we analyze how many sub-networks are needed in a network, and give a recommended number of controllers.

To evaluate whether the recommended number k is the point of diminishing return, we test the best average latencies and the worst-case latencies under different numbers of controllers deployed for all the topologies. The topologies are categorized according to the k. Figs. 4 and 5 show the variation of decreasing ratio at different conditions. The decreasing ratio is defined as $(avglat_{i-1}/avglat_{i})$, where $avglat_{i}$ represents the average latency when i controllers are deployed. The closer to 1 the ratio is, the lower benefit the additional controller brings. In Fig. 4, the ratios of $k = 1$ topologies are closer to 1 after setting the controller number to be more than 1. The ratio of $k = 2$ topology is smaller than 0.8 with two controllers, and is quickly close to 0.9 after setting the controller number to be more than 2. We observe the same phenomena in all categories of topologies. When the number of controllers is smaller than k, the corresponding topology ratio is still relatively small. However, when the number of controllers comes close to k, the ratio becomes relatively close to 1. In Fig. 5, the ratio is defined as $(maxlat_{i-1}/maxlat_{i})$, where the $maxlat_{i}$ represents

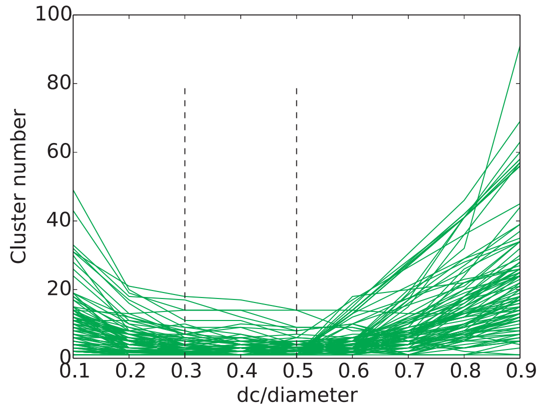


Fig. 6. The number of cluster variation at different d_c.

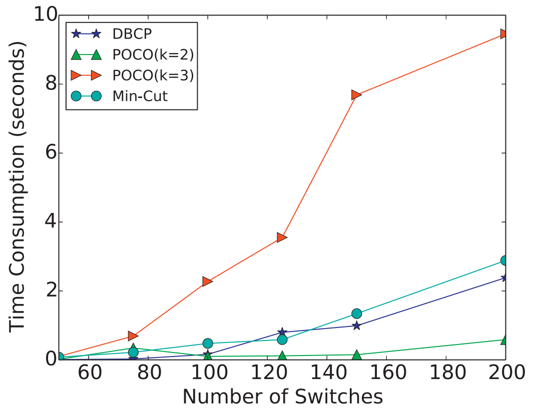


Fig. 7. Comparison on time consumption. The required controller numbers are set as 2 and 3 to POCO respectively.

the worst-case latency when i controllers are deployed. Similar results prove that k is a reasonable value for the controller number in all topologies.

4.2. *Evaluation of* $\mathrm{d_c}$

The value of d_c, the only parameter in our algorithm, affects the clustering performance of DBCP. To estimate the relationship between the clustering performance and d_c, we set d_c to be the values from 1 to the network diameter to compare the performance of DBCP. Fig. 6 shows how the number of clusters varies under different d_c. Each line represents a topology and the cluster number uses our recommended k. The value of k is large when d_c is too large or too small. This is because if d_c is too small, the density of each switch is limited in a short radius, which causes switches to have similar densities and makes the size of the cluster small. If it is too large, the switches density will approximate to the number of all the switches, which also makes the size of cluster much small. When setting d_c from 0.3 times diameter to 0.5 times diameter, the results are relatively constant. So, the optimal value of d_c is from 0.3 times diameter to 0.5 times diameter. Since small d_c decreases the search time, we set it to be 0.3 times diameter in our experiments.

5. Experiment

In order to evaluate the performance of DBCP, various evaluation experiments are carried out. In the following part, we first compare the performance among DBCP, Min-cut [2], and POCO [7] for the CPP problem. The metrics include time consumption, latency, and fault tolerance. Then, we compare the performance between CDBCP and capacity K-center algorithm [5] for the CCPP problem. We focus on the required number of controllers to meet the capacity limitation. We do not compare with Min-cut and POCO, because both of them cannot deal with this problem. The topology set we adopted is the Internet Topology Zoo, which are used in last section.

5.1. *Performance of time consumption*

The time consumption of DBCP can be separated into 4 parts: calculating the density, finding the nearest switch with higher density, clustering, and finding the controller placements. The time complexity of density calculation is affected by d_c, and the number of switches n in a network. The high d_c leads to a high average density $\bar{\rho}$ and high search time, the complexity of this part is calculated as: $O(\bar{\rho}n)$. The time complexity of finding the nearest switch with higher density is approximate to $O(n)$, because most switches are close to the switch with higher density and only several switches are far from the switch with higher density. For the clustering part, DBCP traverses all the nodes and assigns the node to the cluster of its nearest switch with higher density. So the time complexity of clustering is $O(n)$. To select the best controller placement, we need to traverse all the switches within the cluster, the worst-case time complexity is $O(n^2)$. The total time complexity approaches to $O(\bar{\rho}n + n + n + n^2)$.

To compare the time consumption of different methods, we perform DBCP, Min-cut, and POCO at different sizes of network at the same computing platform. The time consumption of DBCP and Min-cut are not changed by varying the controller number k. As shown in Fig. 7, the Min-cut and DBCP have similar result. The time consumption of POCO method is affected by the number of controller dramatically. If only 2 controllers are placed, the response of POCO is very fast. When the number of controller increases, POCO will cost much more time than Min-cut and DBCP. In addition, both Min-cut and POCO need to traverse all the candidate controller numbers to find the optimal one, so DBCP is much faster.

5.2. *Performance of latency*

5.2.1. *Controller-to-switch and inter-controller communication*

Firstly, we study two controller-to-switch latency metrics: average latency and worst-case latency, and one inter-controller latency metric: average inter-controller latency. In the *Biznet* and *UsSignal* networks, the Pareto-optimal curves are shown in Figs. 8 and 9 with 4 and 3 controllers respectively. In these two networks, DBCP and Min-cut approaches achieve better performance than POCO on the controller-to-switch latency. The POCO achieves the best performance on inter-controller latency. This is because DBCP and Min-cut deploy the controllers in different sub-networks, the controllers cannot be deployed as closely as POCO. However, POCO reduces the inter-controller latency by increasing the latency between controllers and switches.

In Fig. 10, we show the results for all the topologies in Topology Zoo on the latency performance between controller and switch. The points with different colors and shapes represent the results of the corresponding approaches. All the topology solutions are shown in the figure. The result of Min-cut is more close to DBCP, but it still cannot exceed the result of DBCP.

5.2.2. *End-to-end communication*

Here, we study the latency of end-to-end communication between different switches. It could be separated into 3 successive parts: the switch-to-controller communication (when a new flow

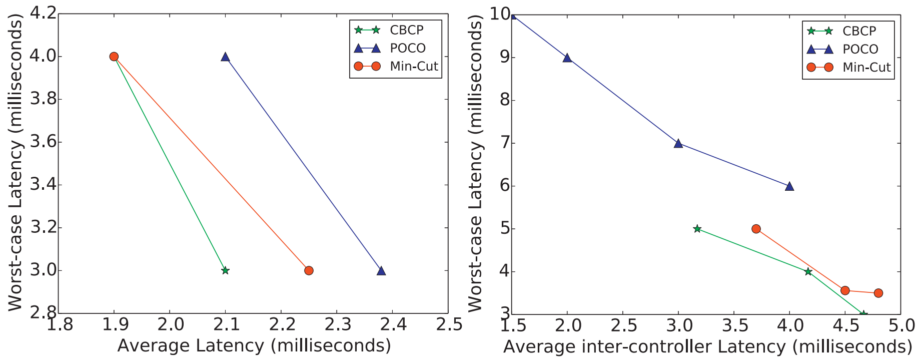


Fig. 8. Pareto-optimal curves of Biznet.

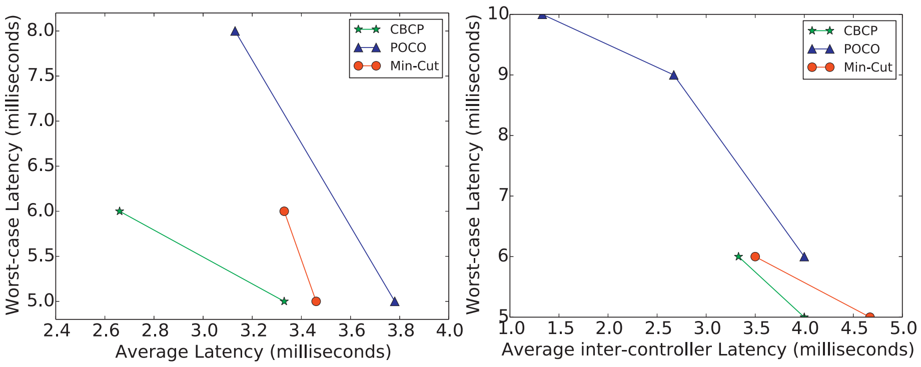


Fig. 9. Pareto-optimal curves of UsSignal.

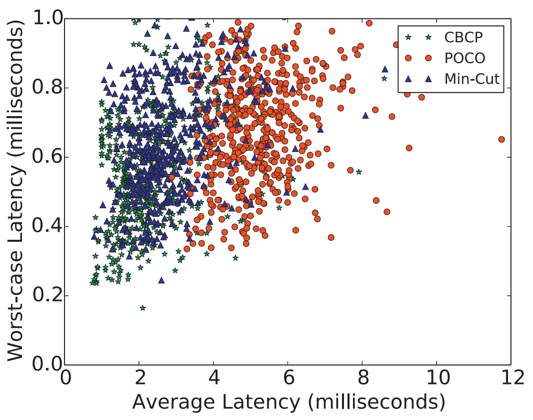


Fig. 10. Comparison on all controller placement solutions. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

comes into a switch, the switch will send a packet-in packet to the controller for instructions), the controller-to-controller communication (it may occur if the flow should go through different sub-networks), and controller-to-switch communication (the controller would send the rule to all related switches). Given the departure switch s_i, which is controlled by controller v_i and destination switch s_j, the switches in the path between them are $Path_{i,j}$. Each switch s_m in $Path_{i,j}$ is controlled by controller v_m and the controllers form the set $V_{i,j}$. The metric to evaluate the latency performance is as follows:

$$m^{latency} = \frac{1}{(|S|-1)|S|} \sum_{s_i,s_j \in S, i \neq j} \times \{d(s_i, v_i) + \max_{s_m \in Path_{i,j}} (d(v_i, v_m) + d(v_m, s_m))\} \quad (14)$$

where S represents all the switches in the network.

The approaches of Min-cut and DBCP both firstly split the network, we use the same objective function $\pi^{latency}$ to find the placements in sub-networks. Given a controller θ and the switches in the sub-network $\mathbb{S}(\theta)$, the objective function of finding the controller placement v is a combination of the $\pi^{avglatency}$, $\pi^{maxlatency}$, and $\pi^{inter-controller}$ according to the proportion of 1:1:1, which is calculated as:

$$\begin{aligned} \pi^{latency} &= \pi^{avglatency} + \pi^{maxlatency} + \pi^{inter-controller} \\ &= \min_{v \in \mathbb{S}(\theta)} \left\{ \frac{1}{|\mathbb{S}(\theta)|} \sum_{s \in \mathbb{S}(\theta)} d(v, s) \right. \\ &\quad \left. + \max_{s \in \mathbb{S}(\theta)} d(v, s) + \frac{1}{|S|} \sum_{s \in (S - \mathbb{S}(\theta))} d(v, s) \right\} \end{aligned} \quad (15)$$

To the heuristic based approach, POCO, we directly use the metric $m^{latency}$ to find the placements. The comparison results are shown in Fig. 11. DBCP achieves the best performance when the network size is larger than 60 switches. When the size of network is less than 60 switches, POCO achieves better performance than DBCP. Due to DBCP and Min-cut cannot deploy controllers as closely as POCO, it decreases the inter-controller latency performance and affects the end-to-end communications. So the performance of POCO is better in small size networks. In a larger size network, the number of candidate placements become extremely large. POCO is a kind of random global search algorithm, which is easy to be trapped into the local optimal solution. Min-cut is an algorithm to find the minimum cut of the network. It does not consider the sub-network structures and some switches in a sub-network may not be very close. Therefore, DBCP is more suitable in large scale networks.

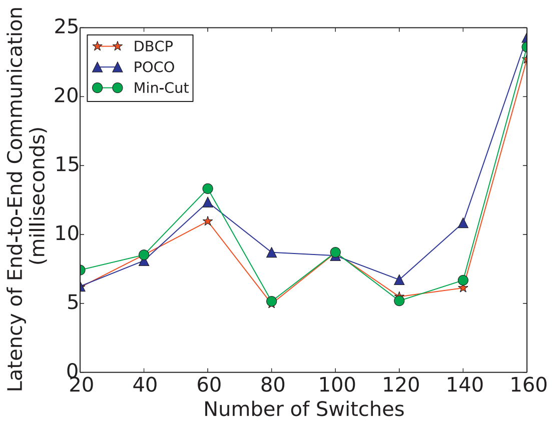


Fig. 11. The comparison of average latency on end-to-end communication based on different approaches.

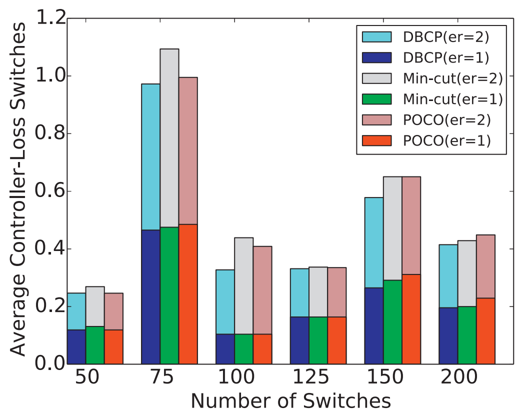


Fig. 12. Average number of controller-loss switches at different sizes of network.

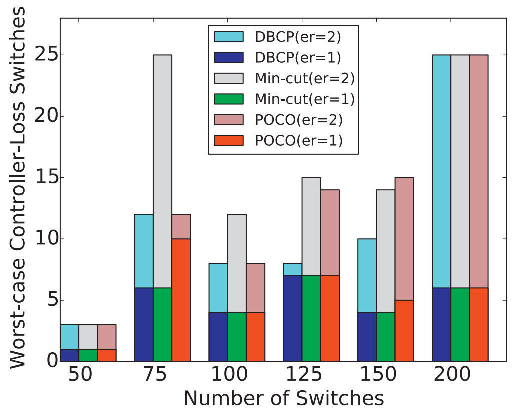


Fig. 13. Worst-case number of controller-loss switches at different sizes of network.

5.3. Performance of fault tolerance

We use the simulations to verify the benefit of DBCP in terms of fault tolerance in Figs. 12 and 13. The network fault may cause some switches to lose their connection to the controllers. The DBCP can decrease the risk for a network.

The Fig. 12 shows the results of different methods under different number of faults. $er = 1$ represents a random link fault occurred in the network. $er = 2$ represents two random link faults occurred. The x-axis is the total number of network nodes, n. The y-axis shows the corresponding $\pi^{avgloss}$. A lower $\pi^{avgloss}$ indicates a better performance of tolerating the network fault. All the methods use the same number of controllers, which is recommended

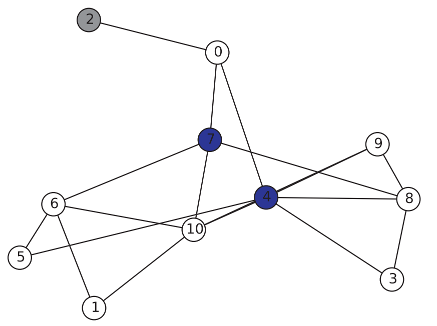


Fig. 14. An example on Sprint topology. (The blue nodes are the recommended locations of controller, the grey node is additional node for increasing fault tolerate performance.). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 3
The performance comparison on adding additional controller.

plan	$er = 1$		$er = 2$	
	$\pi^{avgloss}$	$\pi^{maxloss}$	$\pi^{avgloss}$	$\pi^{maxloss}$
DBCP	0.055	1	0.143	2
DBCP+Random	0.055	1	0.143	2
DBCP+Spoke	0.0	0	0.019	1

by DBCP. Both DBCP and Min-cut use the best average loss and the worst-case loss placement solutions to compare with the best fault tolerate solution of POCO. Generally, DBCP provides the best solutions among different sizes of network. The value of $\pi^{avgloss}$ can be very high even in the relatively small size network (e.g. the 75 switches network). The network structure can also affect the performance of network fault tolerance. As discussed in [2], comparing star, ring, and fat tree structure topologies with the same number of nodes, the fault tolerate performance is very different. The star topology keeps a stable and relatively very low probability of connectivity loss under different sizes.

Different from the average connectivity loss, the worst-case connectivity loss evaluates the worst-case risk of fault in the network. A lower $\pi^{maxloss}$ indicates that the worst-case risk is lower. Fig. 13 shows DBCP provides the best solutions. Sometimes, the performances of these three methods are the same. The $\pi^{maxloss}$ increases with the size of the network and the performance of the network with 75 switches is as bad as that of the network with 200 switches. This indicates the values of $\pi^{maxloss}$ is affected by the size and structure of the networks.

As discussed above, the DBCP provides the best solution to guarantee a relatively better fault tolerate performance. However, if a high quality support network is required, we need to add some additional controllers at some switches with high risk, such as the switches with only one link. For example, a *Sprint* topology is shown in Fig. 14, where each node represents a switch, and the blue nodes are the locations of controllers that are obtained by DBCP. Switch 2 is a switch with high risk since it only has one link, and if a fault occurs at this link, switch 2 can no longer connect to the controllers. So, we add an additional controller at switch 2. The performance after adding the additional controller is shown in Table 3, where "DBCP" represents the original solution. "DBCP+Random" represents the placement after adding a controller at a random place based on the original solution. "DBCP+Spoke" represents the placements after adding a controller at the switch with one link based on the original solution. In this table, we compare the performance under two scenarios: $er = 1$ and $er = 2$. If we randomly add an additional controller, it is useless to improve the fault tolerate performance. On the other hand, if we add a controller on the switch with high risk, the performance is improved

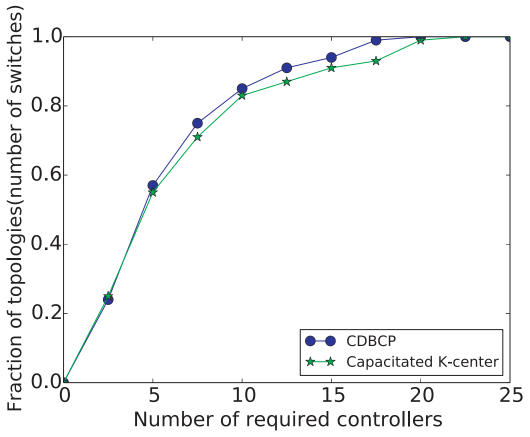


Fig. 15. The optimal number of controller with the limitation of maximum switch number to be controlled by one controller.

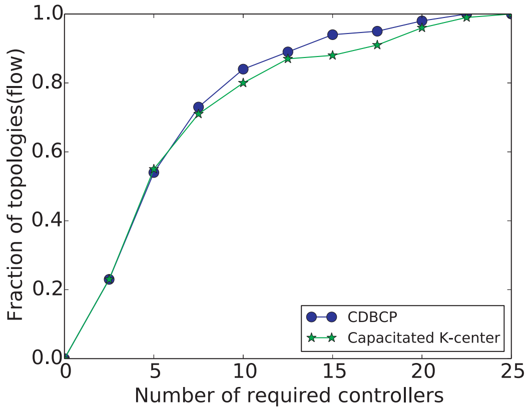


Fig. 16. The optimal number of controller with the limitation of maximum flow to be controlled by one controller.

dramatically. When only one fault occurs, no switch loses its connection to the controllers.

5.4. Performance of capacity controller placement

We compare the CDBCP and capacity K-center algorithm [5] for the capacity controller placement problem. We consider two kinds of scenarios. The first scenario is based on the assumption that the flows are evenly distributed. The capacity is defined as the maximum number of switches that can be controlled, which is calculated as $L(\theta)/l(s)$. The $L(\theta)$ represents the capacity of controller and the $l(s)$ represents the load of controlling a switch. The second scenario is the flows of links are different and 20% of switches are randomly selected as popular destinations. The capacity is defined as the maximum flows that can be controlled, which is constrained as $L(\theta) \geq \sum_{s\in\mathbb{S}(\theta)} l(s)$. The $\mathbb{S}(\theta)$ represents the collection of switches in the sub-network controlled by θ, and $l(s)$ represents the flows of controlling the switch s.

In the experiments, we gradually increase the number of controllers for capacity K-center approach until no overload happens. CDBCP directly uses the recommended number of controllers. We use the fraction of topologies are not overload at the same number of controllers to compare the different performance for the capacity constraint. As the first scenario results illustrated in Fig. 15, to avoid overload, 5 controllers are required using CDBCP approach for 50% of topologies, and at most 20 controllers for all the topologies, whereas using capacity K-center strategy requires 5 and 23 controllers correspondingly. We can see that CDBCP use fewer controllers than capacitated K-center when the topologies require more than 5 controllers. In the second scenario, the performance of CDBCP still better than capacity K-center to the large scale networks. As shown in Fig. 16, to avoid overload, at most 19 controllers are required using CDBCP approaches for all the topologies, whereas using capacity K-center approach requires 22. Therefore, the CDBCP can provide better performance on Capacity Controller Placement of large-scale networks. This is because the capacity K-center algorithm is also one of heuristic based algorithms, and its main idea is to iteratively find a node that is furthest from the current center until k centers meeting the capacity requirement are chosen [14]. The drawback of being easily trapped in the local optimal solution makes its performance worse than that of CDBCP.

6. Discussion and conclusion

In this paper, we discuss the problem of controller placement and propose a general algorithm, named as Density Based Controller Placement (DBCP), which finds the optimal controller number based on the density clustering and then places the controllers. We demonstrate that DBCP has several advantages compared with other methods.

First, it provides the fast response and optimal solution. To obtain the solution of controller placement, DBCP only needs to run the algorithm at one time. In contrary, most conventional approaches obtain the solutions using an iterative method. Thus, for the large-scale network, the conventional approaches are infeasible. DBCP uses the density based clustering to analyze the whole network and separate the network into several internal tightly connected sub-networks, which helps to find the global optimal solution.

Second, it can be implemented for a variety of applications and can be easily extended. DBCP only needs to configure one parameter d_c, thus it can be implemented easily. Also, we can implement DBCP for different applications by changing the distance function and the placement objective functions. In this paper, the distance between two switches is the shortest path hops. In other applications, the distance function can be changed as the delay or physical distance. Using different objective functions, DBCP can provide different solutions to meet the requirements.

Third, it can be easily deployed in the SDN maintenance stage. For a SDN network, a variety of reasons lead to the changes of network structure can occur due to different reasons. We can keep the value of ρ and δ, and update the value of the associated changed nodes to redeploy the networks within little cost.

Additionally, the DBCP full code, which is available at https://github.com/hfsun/DBCP, provides a set of APIs to analyze other networks and a visualization of controller placement solutions.

Acknowledgments

The authors gratefully acknowledge financial support from China Scholarship Council. This research was jointly supported by: (1) National Basic Research Program of China (2013CB329102); (2) National Natural Science Foundation of China (61671079, 61471063, 61421061, 61372120, 61302087, 61372120, 61271019); (3) Beijing Municipal Natural Science Foundation (No. 4152039); (4)Spanish Research Council (No: TIN2013-46883); (5)Regional Government of Madrid (No: S2013/ICE-2894) cofunded by FSE & FEDER.

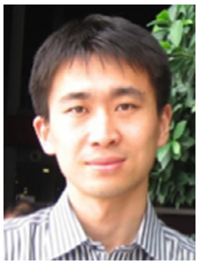
References

[1] F. Hu, Q. Hao, K. Bao, A survey on software-defined network and openflow: from concept to implementation, IEEE Commun. Surv. Tut. 16 (4) (2014) 2181–2206.

[2] Y. Zhang, N. Beheshti, M. Tatipamula, On resilience of split-architecture networks, in: Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE, 2011, pp. 1–6.
[3] B. Heller, R. Sherwood, N. Mckeown, The controller placement problem, Acm Sigcomm Comput. Commun. Rev. 42 (4) (2012) 7–12.
[4] A. Sallahi, M. St-Hilaire, Optimal model for the controller placement problem in software defined networks, IEEE Commun. Lett. 19 (1) (2015) 30–33.
[5] 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.
[6] M.F. Bari, A.R. Roy, S.R. Chowdhury, Q. Zhang, Dynamic controller provisioning in software defined networks, in: 2013 9th International Conference on Network and Service Management (CNSM), 2013, pp. 18–25.
[7] 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. Manage. 12 (1) (2015). 1–1.
[8] A. Jalili, V. Ahmadi, M. Keshtgari, M. Kazemi, Controller placement in software-defined wan using multi objective genetic algorithm, in: 2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI), 2015.
[9] A. Dixit, F. Hao, S. Mukherjee, T.V. Lakshman, R. Kompella, Towards an elastic distributed sdn controller, Acm Sigcomm Comput. Commun. Rev. 43 (4) (2013) 7–12.
[10] R. Alex, L. Alessandro, Machine learning. clustering by fast search and find of density peaks., Science 344 (6191) (2014) 1492–1496.
[11] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, Onix: a distributed control platform for large-scale production networks, in: Usenix Conference on Operating Systems Design and Implementation, 2010, pp. 351–364.
[12] A. Tootoonchian, Y. Ganjali, Hyperflow: a distributed control plane for openflow, in: Internet Network Management Conference on Research on Enterprise NETWORKING, 2010. pp 3–3.
[13] A.R. Curtis, J.C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, Devoflow: scaling flow management for high-performance networks, Acm Sigcomm Comput. Commun. Rev. 41 (4) (2011) 254–265.
[14] F.A. zsoy, M. Pnar, An exact algorithm for the capacitated vertex p-center problem, Comput. Oper. Res. 33 (5) (2006) 1420–1436.
[15] L. Yao, P. Hong, W. Zhang, J. Li, Controller placement and flow based dynamic management problem towards SDN, in: IEEE International Conference on Communication Workshop, 2015.
[16] F.J. Ros, P.M. Ruiz, On reliable controller placements in software-defined networks, Comput. Commun. 77 (2015) 41–51.
[17] T. Erlebach, A. Hall, L. Moonen, A. Panconesi, F. Spieksma, D. Vukadinovi, Robustness of the internet at the topology and routing level, Access Download Stat. 4028 (2006) 260–274.
[18] M. Guo, P. Bhattacharya, Controller placement for improving resilience of software-defined networks, in: International Conference on Networking and Distributed Computing, 2013, pp. 23–27.
[19] A. Clauset, M.E. Newman, C. Moore, Finding community structure in very large networks, Phys. Rev. E 70 (6) (2004) 264–277.
[20] S. Guo, S. Yang, Q. Li, Y. Jiang, Towards controller placement for robust software-defined networks, in: IEEE International PERFORMANCE Computing and Communications Conference, 2015, pp. 1–8.
[21] Y. Hu, W. Wang, X. Gong, X. Que, S. Cheng, Reliability-aware controller placement for software-defined networks, Wireless Communication Over Zigbee for Automotive Inclination Measurement China Communications 11 (2) (2013) 672–675.
[22] M. Steinbrunn, G. Moerkotte, A. Kemper, Heuristic and randomized optimization for the join ordering problem, Vldb J. 6 (3) (1997) 191–208.
[23] P. Czyzak, A. Jaszkiewicz, Pareto simulated annealinga metaheuristic technique for multipleobjective combinatorial optimization, J. Multi-Criteria Decis. Anal. 7 (7) (1998) 34–47.
[24] S. Knight, H. Nguyen, N. Falkner, R. Bowden, M. Roughan, The internet topology zoo, Sel. Areas Commun. IEEE J. 29 (9) (2011) 1765–1775, doi:10.1109/JSAC.2011.111002.

Jianxin Liao obtained his PhD degree at University of Electronics Science and Technology of China in 1996. He is currently the dean of Network Intelligence Research Center and the full professor of State Key laboratory of Networking and Switching Technology in Beijing University of Posts and Telecommunications. He has published hundreds of research papers and several books, and has been granted dozens of patents for inventions. He has won a number of prizes, which include the Premier's Award of Distinguished Young Scientists from National Natural Science Foundation of China in 2005, and the Specially-invited Professor of the "Yangtse River Scholar Award Program" by the China Ministry of Education in 2009. His main creative contributions include mobile intelligent network, service network intelligent, networking architectures and protocols, and multimedia communication. These achievements were conferred the "National Prize for Progress in Science and Technology" twice in 2004 and 2009, respectively.

Haifeng Sun is currently a PhD student in the State Key Laboratory Of Networking And Switching Technology of Beijing University of Posts and Telecommunications. His research interest includes data mining, information retrieval, and Natural Language Processing.



Jingyu Wang obtained his PhD degree from Beijing University of Posts and Telecommunications in 2008. Now he is an associate professor in Beijing University of Posts and Telecommunications, China. His research interests span broad aspects of performance evaluation for Internet and overlay network, consumer electronic, traffic engineering, image/video coding, multimedia communication over wireless network.

Qi Qi obtained his PhD degree from Beijing University of Posts and Telecommunications in 2010. Now she is an assistant professor in Beijing University of Posts and Telecommunications. Her research interests include SIP protocol, communications software, Next Generation Network, Ubiquitous services, and multimedia communication.

Kai Li was born in 1991, obtained his BS degree at Harbin Institute of Technology. He is presently working toward the MS degree at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His research interests include SDN and network virtualization.

Tonghong Li was born in 1968, obtained his PhD degree from Beijing University of Posts and Telecommunications in 1999. He is currently an assistant professor with the department of computer science, Technical University of Madrid, Spain. His main research interests include resource management, distributed system, middleware, wireless net- works, and sensor networks.