Latency-Aware Reliable Controller Placements in SDNs

Yuqi Fan[1(✉)], Yongfeng Xia[1], Weifa Liang[2], and Xiaomin Zhang[1]

[1] School of Computer and Information, Hefei University of Technology, Hefei 230009, Anhui, China
yuqi.fan@hfut.edu.cn

[2] Research School of Computer Science, The Australian National University, Canberra, ACT 0200, Australia

Abstract. Most existing research on controller placement in Software-Defined Networking (SDN) investigated controller placements without jointly taking into account both the communication reliability and the communication latency between controllers and switches if any link in the network fails. In this paper, we first introduce a new latency metric that considers the communication delay between the switches and the controllers with and without the single-link-failure. We then formulate a novel SDN controller placement problem with the aim to minimize the communication delay, for which we propose an efficient algorithm. We also show that there is a non-trivial trade-off between a primary path and its backup path in terms of communication delay. We finally conduct experiments through simulations. Experimental results demonstrate that the proposed algorithm is very promising.

Keywords: SDN · Multiple controller placements · A single link failure · The latency · Placement algorithms

1 Introduction

Software-Defined Networking (SDN) is a new networking paradigm that decouples the control plane from the data plane, making the network management much simpler and flexible [1]. Multiple SDN controllers [2–4] can improve the system performance in terms of scalability, delay, etc. through intelligent controller placements.

Lots of effort on controller placements has been taken in recent years, which focuses mainly on which locations the controllers should be placed, and how to map each switch to one of the placed controllers to minimize the accumulated communication latency between switches and their controllers. For example, Heller et al. [5] tackled the controller placement problem with the aim to minimize the node-to-controller propagation latency, and reduced the problem to the facility location problem. Yao et al. [6] minimized the total cost of flow set-up requests from switches to controllers, where each switch was assigned a

Q. Chen et al. (Eds.): ChinaCom 2016, Part II, LNICST 210, pp. 152–162, 2018.
DOI: 10.1007/978-3-319-66628-0_15

weight that is defined as the product of numbers of flow requests and the delay from each switch to the controller. Yao et al. [7] placed the controllers to the network to minimize the maximum latency under the constraints on controller capacities.

The communication reliability is another important concern in controller placements in SDNs [8]. Hock et al. [9] introduced a resilience framework to cope with the resilience of link outages by proposing a Pareto-based optimal controller placement method. Müller et al. [10] proposed a controller placement strategy to maximize the number of node-disjoint paths between each switch and its controller. Hu et al. [11] introduced a new metric, referred as the expected percentage of control path loss, to maximize the reliability of control networks. Ros and Ruiz [12] proposed a strategy that connects each switch to a subset of controllers instead of a single one to ensure the required reliability.

Notice that a high latency in the backup paths between controllers and switches will degrade the entire network performance. To the best of our knowledge, very little attention in literature has ever been paid on the latency between the failure-free and a single-link-failure cases in SDNs. In this paper, we will deal with the multiple controller placements in an SDN with an objective to minimize the total latency of both primary and backup paths.

The main contributions of this paper are as follows. We investigate the controller placement problem to reduce the latency with and without a single-link-failure. We define a novel metric integrating the latency in both primary and backup paths, and propose an efficient algorithm for multiple controller placements based on the proposed placement metric. We also conduct experiments through simulations to evaluate the performance of the proposed algorithm. Experimental results demonstrate the proposed algorithm is very promising.

The rest of the paper is organized as follows. The network model is introduced, and the problem is precisely defined in Sect. 2. The proposed algorithm is presented in Sect. 3. The performance evaluation of the proposed algorithm is given in Sect. 4, and the conclusions are detailed in Sect. 5.

2 Problem Formulation

We model an SDN network as an undirected graph $G = (V, E)$, where V is the set of switches (or nodes) and E is the set of links. Each controller will be co-located with a switch [9], and each switch is controlled by only one controller. We assume that there is at most one link failure in the network [13]. The notations used in the paper are listed in Table 1.

A primary path $p_{i,k}$ needs to be set up between a switch s_i and its corresponding controller c_k. In case a link in path $p_{i,k}$ fails, a backup path needs to be built to replace the failed primary path. In this paper we aim to find a proper location for each controller, and determine the exact mapping between the controllers and the switches, with the objective to minimize the average accumulated latency between switches and controllers. In other words, our optimization objective is to

Table 1. Table of notations

Notation	Definition
s_i	Node/switch i
(i, j)	The link between nodes i and j
c_k	Controller c_k
C	Controller set
u_k	The processing capacity of controller c_k
$r_{i,k}$	The number of requests from switch s_i to the mapped controller c_k
$x_{i,k}$	Indicate whether switch s_i is mapped to controller c_k $(= 1)$ or not $(= 0)$
$y_{i,k}$	Denotes whether controller c_k is placed onto switch s_i $(= 1)$ or not $(= 0)$
$p_{i,k}$	The link set of the primary path between switch s_i and controller c_k
$l^p_{i,k}$	The latency in the primary path between switch s_i and controller c_k
$l^b_{i,k,i',j'}$	The latency in the backup path between switch s_i and controller c_k under link (i', j') failure
$l_{i,k}$	The accumulated latency in the primary path and backup paths between switch s_i and controller c_k

Minimize:

$$\frac{\sum_{i=1}^{|V|}\sum_{k=1}^{|C|} l_{i,k}}{|S|}. \tag{1}$$

Subject to:

$$l_{i,k} = \lambda_1 l^p_{i,k} + \lambda_2 \frac{\sum_{(i',j')\in p_{i,k}} l^b_{i,k,i',j'}}{|p_{i,k}|}, \tag{2}$$

$$\sum_{k=1}^{|C|} x_{i,k} = 1, \quad \forall s_i \in V, \tag{3}$$

$$\sum_{i=1}^{|V|} y_{i,k} = 1, \quad \forall c_k \in C, \tag{4}$$

$$y_{i,k} \leq x_{i,k}, \quad \forall s_i \in V, \forall c_k \in C, \tag{5}$$

$$\sum_{i=1}^{|V|} x_{i,k} \cdot r_{i,k} \leq u_k, \quad \forall c_k \in C. \tag{6}$$

Equation (2) defines the accumulated latency in the primary path and backup path between a switch s_i and its controller c_k, where parameters λ_1 and $\lambda_2(\lambda_1 + \lambda_2 = 1)$ are constant weights used for weighting between the two latency sources. The latency of the backup path takes into account all possible failures of links in the primary path. Equation (3) ensures that each switch is mapped to exactly

one controller. Equation (4) mandates that each controller is placed onto exactly one switch. Equation (5) dictates that switch s_i is mapped to controller c_k if controller c_k is placed onto switch s_i. Equation (6) signifies that the number of requests from its switches to a controller cannot exceed the computing capacity of the controller.

3 Algorithm for Controller Placements

It is known that the controller placement problem with the aim to minimize the accumulated latency of all primary paths is a $\mathcal{NP}$-hard problem [5]. Clearly, the controller placement problem in this paper is $\mathcal{NP}$-hard too, as the former is a special case of our problem where the accumulated latency of backup paths has not been incorporated into the optimization objective.

In this section, we propose a Latency-Aware Reliable Controller placement algorithm (LARC) for the problem. A switch is mapped to a controller by the shortest path, and the path cost is the total weight of all the links that the path traverses. We create an auxiliary graph with each link cost incorporating both primary path latency and average path latency upon the link failure. We place each new controller on the auxiliary graph by searching the location that incurs the least total path cost between the switches and the controllers. Algorithm LARC consists of two stages.

Stage one: A weighted auxiliary graph, $G' = (V', E')$, is constructed from the SDN G which will be used for the controller placements. Each edge in E' is assigned a weight that is the accumulated latency on both primary and backup paths.

Stage two: Determine the location of each to-be-placed controller and map each switch to one of the controllers, by utilizing the auxiliary graph and the proposed metric for controller placements.

3.1 The Construction of the Auxiliary Graph

The weighted auxiliary graph $G' = (V', E')$ is constructed from the SDN $G = (V, E)$ as follows, where $V' = V$ and $E' = E$. Denote by $w_{i,j}$ the weight of link (i, j). For the failure of link (i, j), we calculate the average path latency $w^f_{i,j}$ between all pairs of nodes. The new weight of link (i, j) in G', $w'_{i,j}$, is calculated by Eq. (7), where weights λ_1 and λ_2 are constants with $\lambda_1 + \lambda_2 = 1$.

$$w'_{i,j} = \lambda_1 w_{i,j} + \lambda_2 w^f_{i,j}. \tag{7}$$

3.2 Controller Placement

The detailed algorithm is given in Algorithm 1, which places the controllers on the network based on the auxiliary graph, and maps each switch to one of the controllers. Specifically, the algorithm proceeds iteratively using the greedy strategy. Within each iteration, a single controller is placed. This procedure continues

until all K controllers are placed. For each controller c_k, Algorithm 1 computes the cost of placing it at a location v, which is the total cost of the shortest paths between all unassigned switches and controller c_k located at v (see steps 3–7 of the algorithm). It then chooses a location v with the minimum cost, and map those unassigned switches to controller c_k one by one until the computing capacity u_k of controller c_k is reached (see steps 8–11 of the algorithm). Once controller c_k has been placed at location v, the path cost of mapping a switch to pre-placed controller may be reduced by mapping the switch to controller c_k. Therefore, we perform a procedure Remap, to change the mapping relationship between all placed controllers and the switches assigned to them.

Algorithm 1. *Controller Placement*

Input: Auxiliary graph $G' = (V', E')$
Input: Set of switches S
Input: Number of controllers K
Output: Set of locations placed with controllers C_p
Output: Mapping relationship between switches and controllers
1: $V_p = V'$,$C_p = \varnothing$,$S' = S$;
2: **for** $k = 1...K$ **do**
3: **for** each location $v \in V_p$ **do**
4: Assume controller c_k is placed at location v;
5: Find the shortest path between each switch $s_i \in S'$ and a controller c_k at location v in G', and assume the cost of the path is $c^p_{i,v}$;
6: Evaluate $c^c_v = \sum_{i \in S'} c^p_{i,v}$ the total cost of the paths from all the switches in S' to controller c_k at location v;
7: **end for**
8: Choose the location v with the least cost c^c_v to place controller c_k;
9: Sort the switches in S' in the non-descending order of path cost $c^p_{i,v}$;
10: Map the switches to controller c_k iteratively provided that the mapping does not exceed the computing capacity u_k of the controller, assuming that S^m is the mapped switch set;
11: $C_p = C_p \bigcup \{v\}$,$S' = S' - S^m$,$V_p = V_p \setminus v$;
12: Remap;
13: Relocate;
14: **end for**

Procedure Remap proceeds iteratively. Within each iteration, for a given switch s_i that has been mapped to a controller, the procedure will calculate the accumulated delay of mapping switch s_i to each controller, and find the controller c_k that incurs the least accumulated delay. Switch s_i is then mapped to controller c_k if this mapping will not exceed the computing capacity of controller c_k.

Changing the location of a placed controller may also reduce the delay between the controllers and switches. Algorithm 1 performs Relocate to further reduce the mapping cost. For each controller c_k that has been placed at a location, Relocate evaluates whether deploying each controller c_k onto an assigned

switch location v will reduce the total delay between the switches and the controllers. The total delay is the cost sum of the minimum cost paths between all assigned switches to controller c_k at location v.

Assume K, N and L are the numbers of controllers, switches and links, respectively. In the first stage of algorithm LARC, to construct the auxiliary graph G' from the network G, we iteratively delete a link in G and calculate the shortest path between all pairs of nodes, using Dijkstra's algorithm, where Dijkstra's algorithm runs in time $O(N^2)$, and hence the calculation of the paths of all pairs of nodes can be performed in $O(N^3)$ time. The construction of the auxiliary graph construction takes $O(L \cdot N^3)$ time.

The second stage of algorithm LARC places the controllers one by one. As the shortest paths has been figured out already in the first stage, the localization of a controller can be determined within $O(N)$ time. Algorithm 1 performs procedures Remap and Relocate after placing a new controller. In the worst case, procedure Remap has to change the mapping relationship between all the K controllers and N switches. The worst case time complexity of procedure Remap is $O(KN)$. For all the placed controllers, procedure Relocate checks all potential locations that can accommodate the controllers, and hence procedure Relocate runs in time $O(KN)$. Algorithm 1 thus takes $O(K^2N)$ time. Consequently, the time complexity of algorithm LARC is $O(L \cdot N^3) + O(K^2N) = O(N^3L + K^2N) = O(N^3L)$ since $N \geq K$.

4 Performance Evaluation

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

4.1 Simulation Setup

We evaluate the proposed algorithm LARC against the state-of-the-arts: SVVR [10] and CPP [5], where algorithm SVVR maximizes the connectivity between switches and controllers, by exploring the path diversity, while algorithm CPP places the controllers in such a way that the average latency from the switches to the controllers is minimized. The network topologies used in the simulation are ATT (ATT North America) and Internet2 [5,14]. The capacities of controllers and the request rates of switches are set as the same as those used by algorithm SVVR. All controllers have identical computing capacity of 1800 kilorequests/s, and each switch generates the requests with the rate of 200 kilorequests/s. We use the geographical distance between two locations as an approximation of latency [5].

4.2 Performance Evaluation of the Proposed Algorithm

4.2.1 Weighting Impact of Backup Paths

We first evaluate the performance of different algorithms LARC, SVVR and CPP by varying the latency weights of primary and backup paths λ_1 and $\lambda_2(\lambda_1+\lambda_2 = 1)$ in Eq. (7), assuming that the number of controllers K is 5.

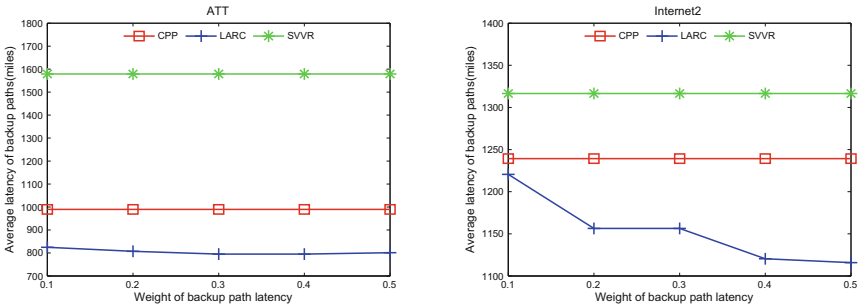

Fig. 1. The average latency of backup paths under different weighting

Figure 1 shows the average latency of backup paths by varying the value of λ_2 after single link failure. Algorithm LARC outperforms algorithm CPP by 24% and 11% for two different networks ATT and Internet2, respectively. Algorithm LARC places the controllers by jointly considering the latencies of both primary and backup paths, while algorithm CPP deploys the controllers with the objective of optimizing the primary path latency only. Algorithm SVVR is the worst one among the three mentioned algorithms. Algorithm SVVR aims to maximize the number of disjoint paths between switches and controllers. However, the latency is not considered in controller placements.

Figure 2 depicts the average latency of primary paths by varying the value of λ_2. Algorithm SVVR has the worst performance, as it focused on finding the maximum number of disjoint paths between switches and controllers for communication reliability without taking into account the communication latency. Algorithm CPP performs better than algorithm LARC on different networks. The average latency of primary paths delivered by algorithm LARC increases from 0.3% to 14% for ATT, and from 0.1% to 17% for Internet2. In general, when the weight λ_2 is no greater than 0.3, algorithm LARC is only slightly worse than CPP, as algorithm LARC considers the latencies of both primary and backup paths, while algorithm CPP places controllers without any consideration of backup path latency.

Figure 3 demonstrates the average accumulated latency of primary and backup paths. The performance of algorithm SVVR is the worst, since it does not consider the latency when deploying the controllers. Algorithm LARC outperforms algorithm CPP by 9% on ATT and 4% on Internet2, respectively.

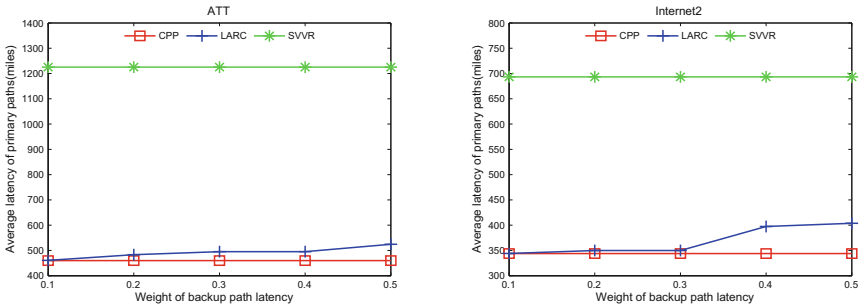

Fig. 2. The average latency of primary paths by varying the latency weight of backup paths

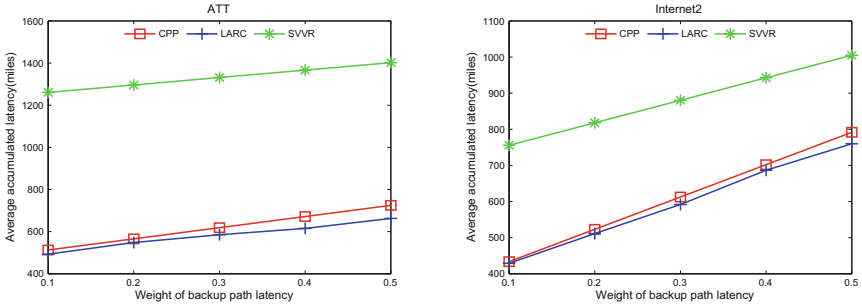

Fig. 3. The average accumulated latency by varying the latency weight of backup paths

4.2.2 Impact of the Number of Controllers

We then study the impact of number of controllers on the performance of different algorithms, assuming the weight λ_2 of backup path latency is set at 0.2. Figure 4 plots the average latency curves of backup paths with different number of controllers. Algorithm LARC outperforms both algorithms SVVR and CPP, since algorithm LARC considers the latencies of both primary and backup paths at the same time. Specifically, with 9 controller placements, algorithm LARC outperforms algorithm CPP in terms of the average delay of backup paths by 27% on ATT and 10% on Internet2, respectively, and the performance improvement diminishes with the growth of the number of controllers.

Figure 5 demonstrates the average latency of primary paths with different number of controllers. Similar to the one shown in Fig. 2, algorithm CPP performs slightly better than algorithm LARC, since algorithm CPP places the controllers with the aim to minimize the latency of primary paths, while algorithm LARC strives for the fine tradeoff of the delays between the primary and backup paths. For both ATT and Internet2, the average latency of primary paths delivered by algorithm CPP is 10% better than that of algorithm LARC, while algorithm SVVR is the worst one.

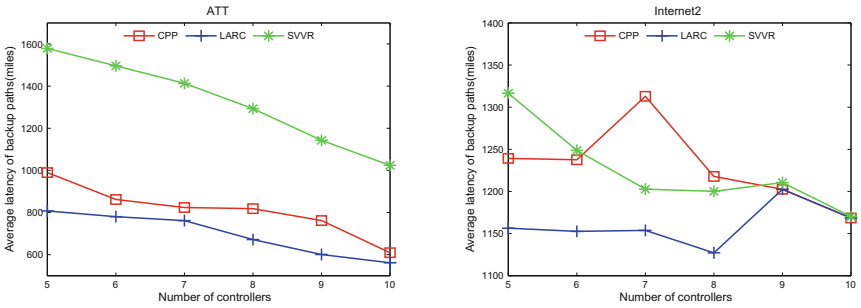

Fig. 4. The average latency of backup paths with different number of controllers

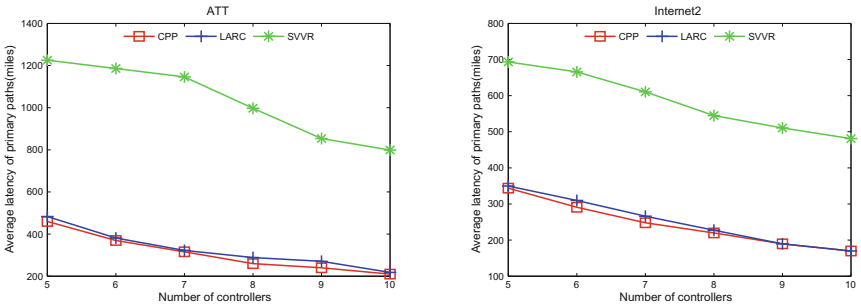

Fig. 5. The average latency of primary paths with different number of controllers

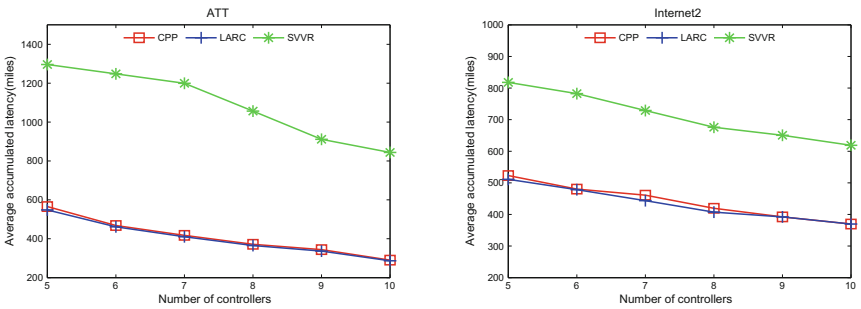

Fig. 6. The average accumulated latency with different number of controllers

Figure 6 illustrates the average accumulated latency by varying the number of controllers. As shown in Fig. 3, algorithm LARC achieves the best performance among the three algorithms, and algorithm LARC outperforms algorithm CPP by 3% on ATT and 4% on Internet2, respectively. Figures 4 and 5 imply that the decrease on the average latency of backup paths can compensate the increase on the average latency of primary paths.

5 Conclusions

Controller placements in Software-Defined Networking (SDN) are crucial in the SDN performance. Most existing studies placed the controllers without jointly considering the communication reliability and the communication latency between controllers and switches if any link in the network fails. In this paper, we introduced a novel latency metric that incorporates the communication delay between the switches and the controllers due to a single link failure. We formulated an SDN controller placement problem with the aim to minimize the average accumulated delay of primary and backup paths between all the switches and their corresponding controllers due to a single link failure, and proposed an efficient algorithm for the problem. We also conducted experiments through simulations. Experimental results demonstrate that the proposed algorithm is very promising.

Acknowledgments. This work was partially supported by Anhui Provincial Natural Science Foundation [1608085MF142].

References

1. Nunes, B.A.A., Mendonca, M., Nguyen, X.N., Obraczka, K., Turletti, T.: A survey of software-defined networking: past, present, and future of programmable networks. IEEE Commun. Surv. Tut. **16**, 1617–1634 (2014)
2. Hassas Yeganeh, S., Ganjali, Y.: Kandoo: a framework for efficient and scalable offloading of control applications. In: ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, pp. 19–24 (2012)
3. Koponen, T., Casado, M., Gude, N., Stribling, J., Poutievski, L., Zhu, M., Ramanathan, R., Iwata, Y., Inoue, H., Hama, T., et al.: Onix: a distributed control platform for large-scale production networks. In: USENIX Symposium on Operating Systems Design and Implementation, vol. 10, pp. 1–6 (2010)
4. Tootoonchian, A., Ganjali, Y.: HyperFlow: a distributed control plane for OpenFlow. In: 2010 Internet Network Management Conference on Research on Enterprise Networking (2010)
5. Heller, B., Sherwood, R., McKeown, N.: The controller placement problem. In: ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, pp. 7–12 (2012)
6. Yao, L., Hong, P., Zhang, W., Li, J., Ni, D.: Controller placement and flow based dynamic management problem towards SDN. In: 2015 IEEE International Conference on Communication Workshop, pp. 363–368 (2015)
7. Yao, G., Bi, J., Li, Y., Guo, L.: On the capacitated controller placement problem in software defined networks. IEEE Commun. Lett. **18**, 1339–1342 (2014)
8. Zhang, Y., Beheshti, N., Tatipamula, M.: On resilience of split-architecture networks. In: 2011 IEEE Global Telecommunications Conference, pp. 1–6 (2011)
9. Hock, D., Hartmann, M., Gebert, S., Jarschel, M., Zinner, T., Tran-Gia, P.: Pareto-optimal resilient controller placement in SDN-based core networks. In: International Teletraffic Congress, pp. 1–9 (2013)
10. Müller, L.F., Oliveira, R.R., Luizelli, M.C., Gaspary, L.P., Barcellos, M.P.: Survivor: an enhanced controller placement strategy for improving SDN survivability. In: 2014 IEEE Global Communications Conference, pp. 1909–1915 (2014)

11. Hu, Y., Wendong, W., Gong, X., Que, X., Shiduan, C.: Reliability-aware controller placement for software-defined networks. In: 2013 IFIP/IEEE International Symposium on Integrated Network Management, pp. 672–675 (2013)
12. Ros, F.J., Ruiz, P.M.: On reliable controller placements in software-defined networks. Comput. Commun. **77**, 41–51 (2016)
13. Markopoulou, A., Iannaccone, G., Bhattacharyya, S., Chuah, C.N., Diot, C.: Characterization of failures in an IP backbone. In: 2004 IEEE International Conference on Computer Communications, vol. 4, pp. 2307–2317 (2004)
14. The Internet Topology Zoo. http://www.topology-zoo.org