Sharing Data Store and Backup Controllers for Resilient Control Plane in Multi-domain SDN

Jiacong Li, Ying Wang, Wenjing Li, Xuesong Qiu
State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications
Beijing, China
Email: {huanshiweiyang, wangy}@bupt.edu.cn

***Abstract*—software-defined networking (SDN) uses a centralized control plane to manage the whole network. If the scale of the network is large, it is necessary to divide it into multiple domains. Since the network scale becomes larger, the probability of failure occurrences is higher. Therefore, it is important to guarantee the control plane resilience in multi-domain SDN. However, the existing approaches cannot store the network state in real time, and do not consider the backup controllers placement problem in multi-domain SDN. In order to ensure the resilience of the control plane in multi-domain SDN, we propose a sharing data store and backup controllers based approach. Sharing data store is used to ensure that each master controller has a view of the whole network and data store can save the network state during the failure time. The sharing backup controllers are used to guarantee the resilience of control plane with minimum cost. Simulations show that our approach can use as less backup controllers as possible to ensure the resilience of control plane.**

Keywords—multi-domain SDN; resilient; control plane; sharing data store; sharing backup controller placement;

I. INTRODUCTION

With the development of the network, the scale of network is larger and larger, so the network management has become a more and more complicated work. To simplify network management, software-defined networking (SDN) [1] is proposed. SDN is a new technology that divides the control plane from the data plane. It relies on a centralized controller that runs on control plane to manage the network. The controller has a view of the whole network, and then calculates the flow tables for each switch, while the switches are just responsible for forwarding the data packets according to the flow tables.

Obviously, the controller plays a significant role in SDN. However, when the scale of the network is large, a controller becomes overloaded and it cannot response to the request from all switches in the required time. One solution of managing large-scale network is splitting it into small parts and each part has a controller. Furthermore, The *OpenFlow* [2] protocol version 1.2 [3] provides a controller role change mechanism to support multiple controllers in multi-domain SDN. Therefore, we can build multi-domain SDN environment to manage the large-scale network based on the role mechanism. However, there are still some problems should be considered. Firstly, it is difficult to keep the consistency of multiple controllers. Since there are several controllers, and each controller only knows the network state of its domain, it is difficult to make each two controllers communicate as soon as the network changes. Moreover, when a controller recovers from a failure, it does not have the current network state of its domain, because this controller does not know the change of the network during its failure time and other controllers do not know either. Additionally, since there are multiple domains in the network, more than one backup controller is needed. So the number and the placement of these backup controllers should be considered.

Numerous previous work has been done in SDN resilience area. Some of them put emphasis on data plane, like [4-6]. Others are related to the consistency guarantees [7], controller replication [8-11], placement of distributed controllers [12-15] and so on. However, most existing approaches to improve the resilience of the control network are focus on controller replication, which needs communication between controllers. Besides few researches have been done about the sharing backup controller placement in multi-domain SDN. Towards addressing this gap, this paper makes the following contributions:

(1) We propose a sharing data store mechanism to guarantee the consistency of controllers in different domains and let the controller, which just recovers from a failure know the present state of the network. Furthermore, we design the data structure to store the state of the network.

(2) We propose a sharing backup controller mechanism in multi-domain SDN to increase the resiliency of the large-scale network. To make the performance of network optimal with using as less backup controllers as possible, we calculate the number of backup controllers based on probabilities and use multi-objective optimization algorithm to compute the placement scheme of backup controllers.

(3) We implement and validate our mechanism in a controlled environment. We conduct simulations to show the validity of the approach, and evaluate the performance of the approach in networks of different scales and probabilities of failure occurrences.

The rest of this paper is organized as follows. Section II discusses the related work. Section III describes the controller consistency and backup controller placement problem. Section IV describes the sharing data store and backup controller mechanisms. Section V describes our experimental methodology and evaluates the performance of our approach, and this paper is concluded in Section VI.

II. Related Work

In order to improve the resilience and reliability of network. Many researchers have paid much attention to the controller replication [8-11] and controller placement problem [12-15].

In general, controller replication approaches can be classified into two categories: the active replication approaches and the passive replication approaches. Paulo Fonseca et al. [8] discussed these two replication approaches. In the case of active replication, the switch connects with multiple controllers that process the request. In passive replication, the switch connects with only one controller that processes the requests and updates the other controllers. Eros et al. [9-10] explored the OpenFlow roles for the design of resilient SDN architectures relying on multi-controllers. In the case of active replication, all controllers play the *equal* role and they process the request at the same time. In passive replication, only one of controllers plays the *master* role, the others play the *slave* role. All the switches connect with the *master* controller, and the *master* controller processes the requests from the switches, then it sends the updates to other *slave* controllers. Besides these two replication approaches, Paulo Fonseca et al. [11] described a novel mechanism that provides an increase of resilience in SDN using a *CPRecovery* component organization.

However, each replication approach has its own disadvantage. In active replication, it is difficult to guarantee the totally ordered delivery of all messages to all controllers; all controllers keep the entire network view, which may be undesirable and so on. In order to solve these problems, Eros et al. [9-10] implemented a strategy of active replication in the *Ryu* controller, using the *OpenReplica* service to ensure consistent state among the distributed controllers. In passive replication, the slaves save the processing costs, while it requires all *slave* controllers to monitor the *master* to guarantee the consistency in case of failures. The replication component proposed by Paulo Fonseca et al. [11] can replicate the network state to the backup controller from the primary controller, but this approach does not consider that replication of the network state after the primary controller breaks down.

The controller placement problem [12-15] can be classified into two categories: the propagation delay based and the network reliability based. The propagation delay based approaches [12] found the location of the controller and make the total delay minimum with the least number of controllers. Jimenez et al. [12] defined a metric to evaluate the candidate nodes that satisfy the required delay. The network reliability based [13-14] approaches find the placement of controllers to keep the network reliable with using as less controllers as possible. Hu et al. [13] used the expected percentage of control path loss to characterize the reliability of SDN control networks. Muller et al. [14] formulated the problem as a binary integer programming to maximize the average number of disjoint paths between devices and controllers.

In order to improve the reliability of control network and satisfy the required propagation delay at the same time, Qinghong Zhong et al. [15] proposed a min-cover based controller placement approach. They first proposed two reliability metrics, which consider how many switches may lose connection to controllers in case of a single-link failure. Then they defined the neighborhood of a vertex and the min-cover of a network, based on which they give the formulation of controller placement problem. They also proposed a heuristic method to find the min-cover to improve the efficiency of calculating the controller placement solution.

However, few researches about backup controller placement problem in multi-domain SDN have been done, and this paper focuses on this problem. In order to keep the consistency of multiple controllers, we propose a sharing data store, to save the network state at any time. Additionally, we propose a sharing backup controller placement approach in multi-domain SDN, which can improve the control network reliability while meeting the required delay.

III. The Proposed Resilient Control Plane Design and Implementation

In this section, we first introduce the architecture of multi-domain SDN with sharing data store and describe the process of failure recovery and failure repair with sharing data store. Then, we provide a sharing backup controller mechanism. Firstly, we calculate the number of backup controllers based on probabilities. Then we propose an algorithm to find the backup controllers placement scheme in multi-domain SDN efficiently.

A. The Sharing Datastore

In order to keep the consistency of the control plane without the communication of controllers, and increase the resilience of the whole network at the same time, we propose a sharing data store to save the whole network state at any time.

Architecture Design. There are multiple domains in network and each domain has a controller, which acts as *master*. In order to make sure each *master* controller has information of the whole network, we design a sharing data store. The architecture is shown as Fig. 1.

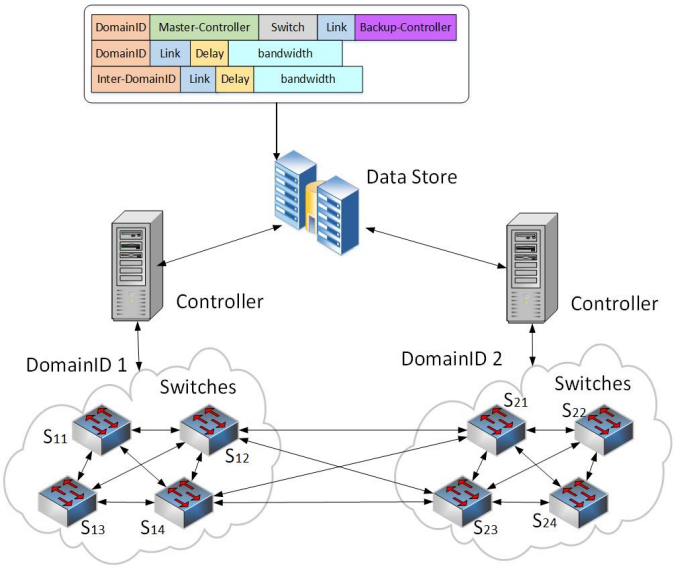


Fig. 1. Architecture Design

As Fig.1 shows, the data store has all topologies of each domain, which means the data store knows the whole network state. When a master controller needs the whole network

information, it sends a request message to the data store, and then the data store replies this request with the whole network information. When a master controller finds that its network has changed, the controller updates the information of the new topology in data store as soon as possible to ensure the data store has the topology of the current network. There is a *calculator* module in data store, which can be used to compute the placement of backup controllers according to the data of topology. After calculating the backup controllers of each domain, data store sends the results to master controllers, and then controllers send the results to every switch, to make sure each switch knows the backup controller.

Failure Recovery. After the data store knows the topology of the whole network, the *calculator* module will calculate the backup controllers and send the results to *master* controllers in each domain, and then the *master* controllers send the information to their switches. Therefore, each switch has a list of backup controllers. The process of failure recovery can be summarized by the following steps:

1) As soon as a switch detects the controller failure in domain *A*, this switch looks for the first backup controller *C* in its backup controller list, and verifies if the controller *C* is available. Then informs the controller *C* that controller *C* needs to become the new *master* controller of domain *A*.

2) After controller *C* receives the message that controller *C* needs to become a master controller in domain *A*, it sends a request message to the data store to get the switches and the network state in domain *A*, then it sends a *role-request* message to each switch in domain *A*.

3) Each switch in domain *A* replies the controller *C* with *role-reply* message after it receives the *role-request* message and changes its *master* controller. From now on, all switches in domain *A* have a new controller.

Failure Repair. This occurs when the faulty controller becomes alive and can be *master* again. The faulty controller is available, but it does not know the network state of domain *A* during its failure time, so it cannot control the switches in domain *A* directly, or it needs to get the network state from the switches again. In order to avoid unnecessary cost, we use sharing data store to save the network state. When the controller becomes alive again, it inquires the current *master* controller of domain *A* in data store and gets the current network state from the data store.

Fig. 2 depicts the process of failure repair. In this case, controller *C′* was the master of domain *A*, but it failed. Then controller *C* becomes the new master as previous mentioned. When the controller *C′* is available again, the following steps are executed:

Phase 1. Controller *C′* sends a *domainID-request* message to data store to get the current network state and database sends the information of its domain.

Phase 2. Controller *C′* sends a *start-migration* message to controller *C* to start the migration process. Then controller C sends a *barrier-request* message to switches to interrupt these switches sending requests. The switches reply this request with a *barrier-reply* message and stop sending messages. After controller *C* processes the request received before, it sends a *flow-mod* message to switches, then switches reply it with a *flow-mod-reply* message. As soon as controller *C* has processed all the requests, it sends an *end-migration* message to controller *C′*.

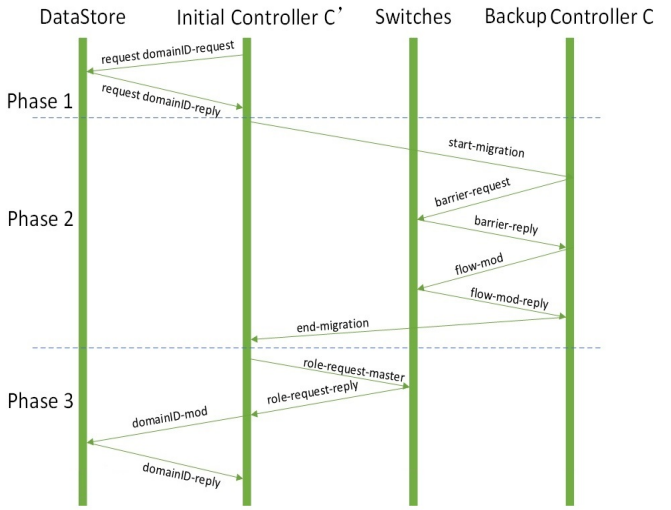


Fig. 2. Process Failure Repair

Phase 3. Controller *C′* sends a *role-request-master* message to switches to change their master controller, and then the switches reply with a *role-request-reply* message. At last, controller *C′* sends a *domainID-mod* message to data store to save the new network state, and then data store confirms this message with a *domainID-reply* message.

B. Implementation of Sharing Backup Controller Placement

In order to construct an effective and reliable control network in multiple domain SDN, we propose a sharing backup controller method. Sharing backup controller means these backup controllers may be used by multiple domains. There are three main questions to be considered: how many backup controllers are needed in the network, which domains are these backup controllers responsible for and where to place them. Many researches [12-15] have been done in controller placement area. However, they mainly put emphasis on the master controller placement in single domain SDN. In this paper, we focus on backup controller placement in multiple domains. The sharing backup controller placement can be described as follows:

Input. Tuple $I = \{G(N, L); C; D; fault_c; delay_G\}$ represents the input of the problem. The physical topology is denoted by an undirected graph *G = (N, L)*, where *N* denotes the set of all the switches, *L* represents the set of all the links between the switches. $C = \{S_{ij}\}$ represents the set of master controllers, where S_{ij} denotes the switch *j* in the domain *i*. The set of domains is given by *D*. $fault_c$ gives the probability of controller failure. $delay_G$ represents the latency of each two adjacent nodes.

Output. The tuple $O = \{N^{BC}; k\}$ represents the output of our implement. The backup controller placements are denoted

by N^{BC}. It is a subset of N. The number of backup controllers is represented by k.

Objective. There are three goals of the proposed strategy. The first one is to make sure the cost minimum, in other words, to make the number of backup controllers minimum. This goal is modeled as function (1), where x_{ij} means whether the backup controller connect to the switch S_{ij}, if it connects to the switch, $x_{ij} = 1$, otherwise $x_{ij} = 0$. S_{ij} means the switch j in the domain i. The range of i is from 1 to the number of domains $|D|$ and N means the set of switches.

$$\min \sum_{1<i<|D|, j\in N} x_{ij} \tag{1}$$

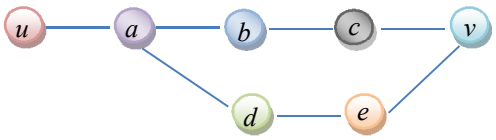
The second one is to make the total delay minimum, as function (2) shows, where S_{xy} represents the backup controller placement.

$$\min \sum_{1<i<|D| \,\&\, i\neq x, j\in N} delay(S_{ij}, S_{xy}) \tag{2}$$

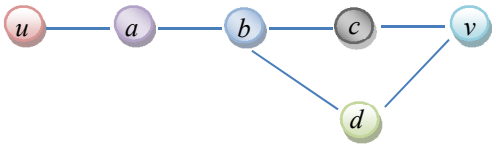
Lastly, we should consider the reliability of the network. We propose a metric *reliability factor* based on these metrics. First, we find that taking overlapping link as a factor is not complete. An instance shown in Fig.3, (a), (b) and (c). In (a), there are two paths (u, a, b, c, v) and (u, a, d, e, v) between node u and v. The amount of paths is 2, the average length of paths is 4, and maximum rate of overlapping link is 1. While in (b), there also have two paths between node u and v. The number of paths is also 2, the average length of paths is 4, and the maximum rate of overlapping link is 1. Obviously, (a) is more reliable than (b). Based on this, we define the average correlation of all paths between two nodes as follows:

$$cor_{u,v} = \frac{\sum_{l\in L}\sum_{i=1}^{n_{u,v}} X_{l,P_{u,v,i}}}{n_{u,v,l}} / n_{u,v} \tag{3}$$

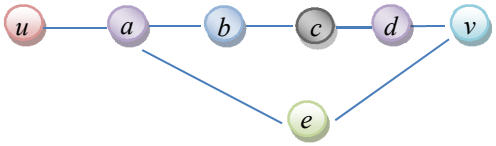
Where $X_{l,P_{u,v,i}}$ denotes the link l whether belongs to the path i between node u and v, and $n_{u,v,l}$ represents the number of links on all paths between u and v. We can observe that the lower the average correlation is, the higher reliability is.



(a)



(b)



(c)

Fig. 3. The different paths between node u and v

In addition, when the average length, the amount of paths and average correlation are equal between two node pairs, like (a) and (c) in Fig. 3. The average length of their paths is 4, the amount of paths is 2, and the average correlation is 4/7. But (c) is more reliable than (a). Because the differences in length of two paths in (c) is larger than that in (a). Therefore, we define the difference in length of all paths as follows:

$$dif_{u,v} = \frac{\frac{\sum_{i=1}^{n_{u,v}} \left(len_{u,v,i} - \overline{len_{u,v}}\right)^2}{\overline{len_{u,v}}}}{n_{u,v}} \tag{4}$$

Where $\overline{len_{u,v}}$ denotes the average length between node u and v as formula (5) shows.

$$\overline{len_{u,v}} = \frac{\sum_{i=1}^{n_{u,v}} len_{u,v,i}}{n_{u,v}} \tag{5}$$

Based on the metric mentioned above, we proposed *reliability factor* to represent the reliability between the two nodes in network as equation (6) shows.

$$rf_{u,v} = \frac{n_{u,v}}{\overline{len_{u,v}} * cor_{u,v} * (1 - dif_{u,v})} \tag{6}$$

The reliability of the whole network can be represented as formula (7) shows, where S_{xy} represents the backup controller placement and k means the number of backup controllers.

$$\max\left(\frac{1}{k}\right) \sum_{v\in N} rf_{S_{xy},v} \tag{7}$$

Constraints. The constrains of our model can be divided into two categories: *placement-related* and *cost-related*.

The first three constraints (8 are *placement-related*. They ensure correctness for the placement of controller instances in the topology. Constraint (8) guarantees that the backup controller is not placed on the master controller node.

$$S_{ij} \notin C, \forall x_{ij} = 1 \tag{8}$$

The *cost-related* constraint (12) ensures that the number of backup controllers is in a certain range based on probability. For a possible fault F_i, if it happens we denoted $F_i = 1$ and $F_i = 0$ otherwise. It is obvious the random variable F_i is with the Bernoulli distribution. The number of failures can be represented by equation (9) and the probability of k' controllers fail at the same time is as equation (10) shows. Formula (11) means the probability of k' concurrent controller fault occurrences is less than a certain value to guarantee the number of backup controllers is enough to responsible for the whole network.

$$|F| = \sum F_i \tag{9}$$

$$p(|F| = k') = \binom{|C|}{k} f_c^{\,k'} (1 - f_c)^{|C|-k'} \tag{10}$$

$$p(|F| = k') < \varepsilon \tag{11}$$

$$k' \leq k \leq |C| \tag{12}$$

C. Algorithm design of Sharing Backup Controller Placement

Since there are three optimization objectives in our model, we use Particle Swarm Optimization (PSO) algorithm to solve our problem. PSO algorithm is proposed by Kennedy J et al. [16].

In this algorithm, the entity is abstracted as particles, and the position of particle is the solution of the problem. PSO uses an archive to save the non-dominated set during searching process, and uses adaptive mesh method to choose global guidance from the archive. Additionally, in order to enhance the local search ability of this algorithm, we use mutation operator.

This algorithm mainly has three parts: the archive and pruning of non-dominated set, global guidance strategies and keeping diversity. The archive and pruning of non-dominated set is used to keep the elite solution and control the quantity of solutions, the global guidance strategies are used to control the direction and speed of particle swarm evolutionary, and the diversity keeping is used to find global optimal solution based on genetic algorithm.

The procedure of calculating the position of sharing backup controller placement is shown in algorithm 1, where *PopSize* means the size of particle swarm, *X* denotes the position matrix of particle, *V* represents the speed matrix, *F* denotes the faulty probability matrix, *pm* means the probability of mutation and *pc* means the probability of crossover. We get these information from input tuple. P_{best} represents the output tuple.

Algorithm 1 Algorithm of SBC

Input: *PopSize*, *X*, *V*, *F*, *L*
Output: P_{best}
Procedure:
1: *loop*=0;
2: (*X*,*V*)=*InitPop*(*PopSize*);
3: *FN* = *EvaluateFitness*(*X*,*L*,*F*);
4: *ND* = *getNonDominatedResult*(*X*,*FN*);
5: *Archive* = *saveToArchive*(*X*,*Archive*);
6: $P_{best} = X$;
7: $g_{best} = getGBest(Archive)$;
8: **while** (*loop* < *MaxLoop*) **do**
9: *loop* = *loop* + 1;
10: *V* = *UpdateSpeed*(*X*,*V*);
11: *X* = *UpdatePosition*(*X*,*V*);
12: *FN* = *EvaluateFitness*(*X*,*L*,*F*);
13: *CO* = *crossover*(*X*,*Archive*,*pc*);
14: *TEMP* = *CO* + *X*;
15: *FNT* = *EvaluateFitness*(*TEMP*,*L*,*F*);
16: *ND* = *getNonDominatedResult*(*TEMP*,*FNT*);
17: *Archive* = *saveToArchive*(*ND*,*Archive*);
18: *X* = *mutate*(*X*,*pm*);
19: $P_{best} = updatePBest(P_{best}, X)$;
20: $g_{best} = getGBest(Archive)$;
21: **end while**

IV. Performance Evaluation

In this section, we conduct simulations to verify the validity of our approach. We first introduce our experimental environment, and then we discuss the metrics we used to evaluate our approach. Last, we show the results of the evaluation performance of our method, and the results of the comparison with other approaches.

A. Experimental Environment

We use the network topology from Brite topology generator [17], which can generate multi-domain network randomly. We assume that the failure probability of each link is the same, and the propagation delay of each edge in the same domain is lower than the latency of edges across different domains. Additionally, for simplicity, we assume that there are 10 nodes in each domain.

We evaluate our approach in two aspects. First, we evaluate the performance of our approach. We compare the number of backup controllers and the average responsibility factor with the approach, which designs backup controllers for each domain, denoted as *BED*. Second, we compare our approach with the existing approaches of controller placement. One approach is based on the shortest path, that means this method put the backup controllers where can make the paths shortest between the backup controller and switches. This approach is denoted as *Shortest*. The other approach put the backup controllers randomly, without considering the delay and reliability of the network, which denoted as *Random*.

B. Evaluation Metrics

In this evaluation, we use two metrics to compare our approach with *BED*. One is the number of backup controllers in different network scales. The other one is the average reliability factor, as formula (7) shows, to represent the reliability of the network.

C. The Results of Evaluating Performance

We evaluate our approach in networks of different scales and failure probabilities. We generate networks of 1, 3, 5, 7, 9, and 11 domains randomly and each domain has 10 nodes. The results are shown in Fig.4 and Fig.5. When the number of domains increases, the number of needed backup controllers increases too. We also find that as the faulty probability of master controller gets larger, more backup controllers are needed to manage the whole network. Because when the faulty probability of master controller gets larger, more controllers will get down, more backup controllers are needed to cover the whole network.

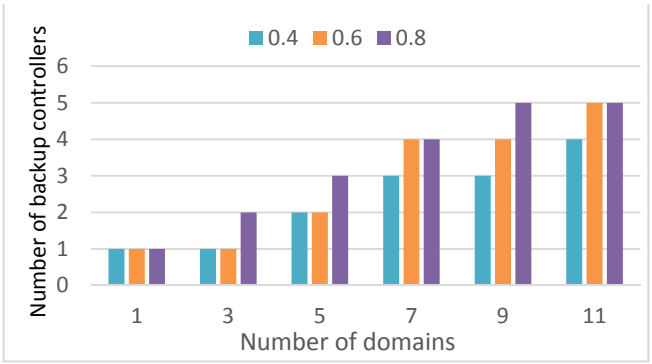


Fig. 4. The number of backup controllers in different networks

Additionally, we can find with the increasing of the domain number, the reliability of the network gets lower. It also can be seen in Fig. 5 that the larger the faulty probability of master controller is, the lower the reliability is.

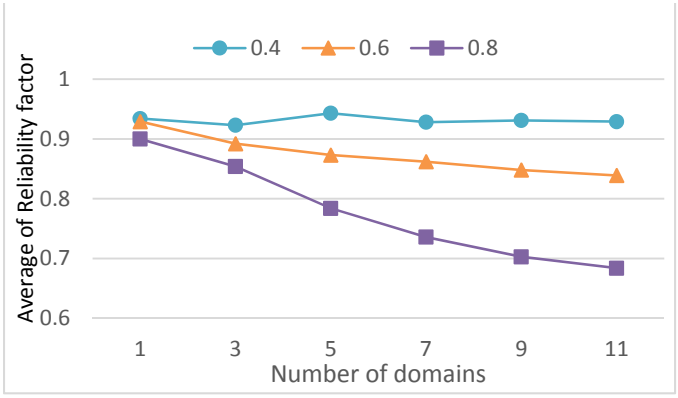


Fig. 5. The reliability of different networks

D. *The Results of Comparision With Other Approaches*

Fig. 6 shows the result of the proposed approach compared with *Shortest* and *Random* approach. We can find that reliability of our approach is higher than other approaches and has a steadily good performance in networks of different link failure probability.

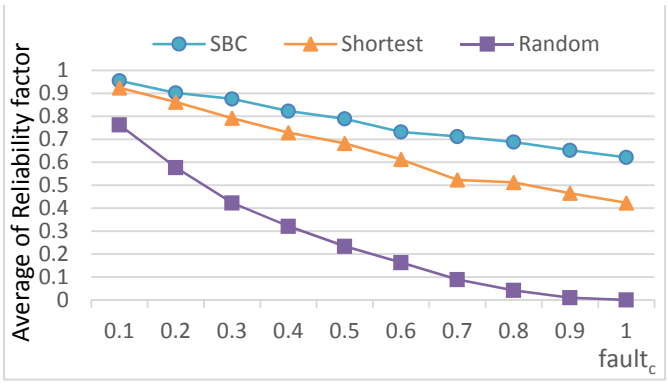


Fig. 6. The reliability factor of different approaches with different $fault_c$

Then we compare these approaches based on metric reliability factor in different number of domains. We generate networks of 3, 5, 7, 9, and 11 domains randomly and each domain has 10 nodes, and the faulty probability of master controller is 0.4. Then we observe the reliability of the network based on the metric *reliability factor*. The result is given by Fig. 7. We can find that reliability of our approach is higher than other approaches and has steadily good performance in networks of different scales.

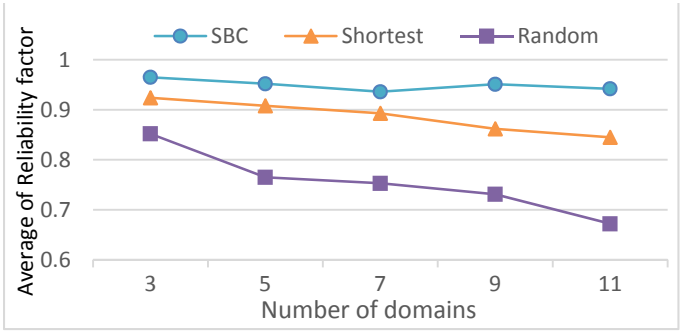


Fig. 7. The reliability factor of different approaches in different scales

Last, we compare our approach with *BED* based on the number of backup controller and the reliability in the network of different scales. We generate networks of 1, 3, 5, 7, 9, and 11 domains randomly, each domain has 10 nodes, and the faulty probability of master controller is 0.4. Fig. 8 shows the number of backup controllers in different approaches. We can find the number of backup controllers our approach needs is less than single backup approach. Fig. 9 shows the reliability of these two approaches. We can find that reliability of our approach is nearly equal to the *BED* approach.

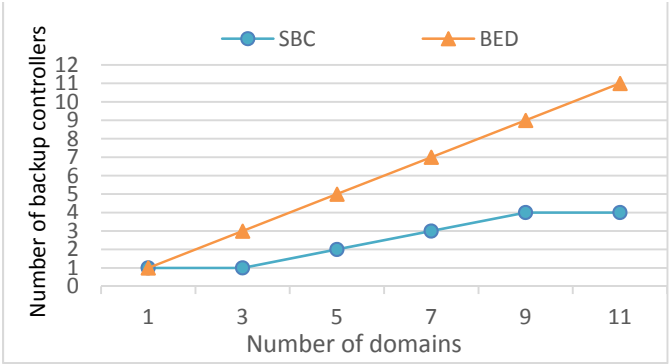


Fig. 8. The number of backup controllers in different approaches

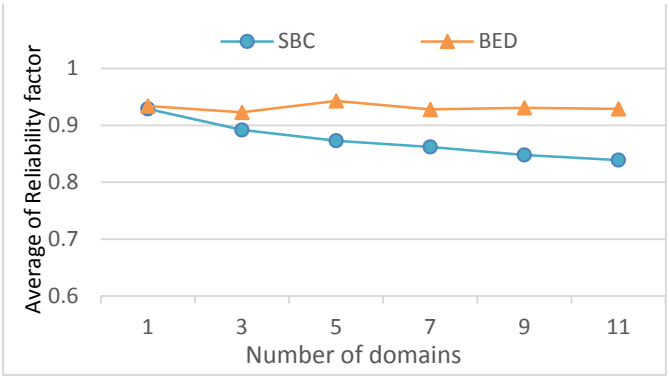


Fig. 9. The reliability of different approaches

V. CONCLUSION

In this paper, we have investigated the failure recovery problem and the backup controller placement problem in multi-domain SDN. In order to improve the resilience and reliability of control network in multi-domain SDN, we propose a sharing data store and backup controllers based approach. We first design the data store to save the network state for failure recovery without communication between two controllers. Then we discuss how to decide the number and the placement of backup controllers. We calculate the number of the backup controllers based on probability. In order to find the backup controller placement to improve the efficiency of calculating we also propose a method, which combines PSO and genetic algorithm. Simulations show that our approach can ensure the control network resilience and reliability in multi-domain SDN environment.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (61501044).

REFERENCES

[1] Hyojoon Kim, Feamster N. “Improving network management with software defined networking.” In: Communications Magazine, IEEE. 2013, 51(2), pp.114-119.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, and J. S. Turner, “Openflow: enabling innovation in campus networks,” Computer Communication Review, vol. 38, no. 2, pp. 69–74, 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746.

[3] O. N. Foundation. (2011) Openflow switch specification version 1.2.0 (wire protocol 0x04). Website. https://www.opennetworking.org/images/stories/downloads/sdnresources/

[4] Sgambelluri A, Giorgetti A, Cugini F, et al. OpenFlow-Based Segment Protection in Ethernet Networks[J]. Journal of Optical Communications & Networking, 2013, 5(9):1066-1075.

[5] Capone, A, Cascone, C, Nguyen, A.Q.T, et al. Detour planning for fast and reliable failure recovery in SDN with OpenState[C]// Design of Reliable Communication Networks. IEEE, 2014:25-32.

[6] Cascone C, Pollini L, Sanvito D, et al. SPIDER: Fault Resilient SDN Pipeline with Recovery Delay Guarantees[J]. Computer Science, 2015.

[7] F. A. Botelho, F. M. V. Ramos, D. Kreutz, and A. N. Bessani, “On the feasibility of a consistent and fault-tolerant data store for sdns,” in Software Defined Networks (EWSDN), 2013 Second European Workshop on. IEEE, 2013, pp. 38–43.

[8] Fonseca P, Bennesby R, Mota E, et al. Resilience of SDNs based On active and passive replication mechanisms[C]// IEEE Global Communications Conference. 2013:2188-2193.

[9] Spalla E S, Mafioletti D R, Liberato A B, et al. Resilient Strategies to SDN: An Approach Focused on Actively Replicated Controllers[C]// Brazilian Symposium on Computer Networks & Distributed Systems. IEEE, 2015:246-259.

[10] Spalla E S, Mafioletti D R, Liberato A B, et al. AR2C2: Actively replicated controllers for SDN resilient control plane[C]// NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium. 2016.

[11] Fonseca, P, Bennesby, R, Mota, E, et al. A replication component for resilient OpenFlow-based networking[C]// Network Operations and Management Symposium. IEEE, 2012:933-939.

[12] Jimenez, Y. Cervello-Pastor, C. Garcia, A. J. “On the controller placement for designing distributed SDN control layer.” In: Networking Conference, IFIP. Trondheim, 2-4 June 2014, pp. 1-9.

[13] Yannan Hu, Wendong Wang, Xiangyang Gong, Xirong Que, Shiduan Cheng. “On reliability-optimized controller placement for Software-Defined Networks.” In: Communications, China, vol. 11(2), Feb. 2014, pp. 38-54.

[14] Muller, L.F; Oliveira, R. R. Luizelli, M. C. “Survivor: An enhanced controller placement strategy for improving SDN survivability.” In: GLOBECOM , Austin, TX. 8-12 Dec. 2014, pp. 1909-1915.

[15] Qinghong Zhong, Ying Wang, Wenjing Li, et al. A min-cover based controller placement approach to build reliable control network in SDN[C]// NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium. 2016.

[16] Kennedy J，Eberhart R C．Particle swarlll optimization．Proceedings of International Conference on Neural Networks，Perth，Australia，1995：1942-1948.

[17] [Online]. Available: http://www.cs.bu.edu/brite/