Telecommunication Systems (2021) 77:377–388
https://doi.org/10.1007/s11235-021-00768-z

An integrated multi-controller management framework for highly reliable software defined networking

Yao-ying Tzeng[1] · Chung-An Shen[1]

Accepted: 13 February 2021 / Published online: 3 March 2021

Abstract
Software-defined networking (SDN) has become the technology of choice for designing the next-generation network infrastructure that is featured with high-volume traffics, rapidly increased scale, and dynamic nature. Furthermore, to deploy multiple controllers in the control plane of SDN is widely considered with the aim of improving the stability and reliability of the network. This paper presents an integrated framework for a comprehensive multi-controller management in SDN. The proposed framework is comprised of a network planning phase and a runtime maintenance phase. Novel algorithms are proposed in the network planning phase to estimate the required number of controllers in the network, to determine the nodes for placing the controllers, and to assign the switch to its managing controller. Moreover, these algorithms are designed by mitigating the problems of device isolation and controller overload such that the reliability and stability of the control plane can be enhanced. In addition, a mechanism based on the State Behavior Tree is proposed in the runtime maintenance phase of the framework. This mechanism dynamically manages the loading of the controller during the execution time so that the occurrence of the controller overload is minimized. The experimental results show that, compared to the prior arts, the proposed framework reduces the isolation probability by up to 89% and increases the device connectivity by up to 34%. The occurrence of the controller overload during runtime is also significantly decreased.

Keywords Software-defined networking (SDN) · Multi-controller · Controller placement · Recovery · Reliability · Behavior tree

1 Introduction

Software-defined networking (SDN) has become the technology of choice for designing the next-generation network infrastructure that is featured with high-volume traffics and dynamic nature [1]. The basic idea of SDN is that the network management, commonly known as the control plane, is decoupled from the forwarding functions, known as the data plane [1]. Furthermore, the control plane is usually centralized to a SDN controller for managing the entire network [1, 2]. This concept of data/control decoupling and the utilization of the centralized controller enable a programmable management of the network where the underlying infrastructure is abstracted from high-level applications. However, with the rapidly increased network scale and data traffic, to manage the entire network by relying only on one SDN controller could jeopardize the reliability, stability, and scalability of the network. For example, the centralized SDN controller could face a Single Point of Failure (SPOF) problem [2] resulting in the disconnections between the control plane and the data plane. In addition, a malfunction of the controller machine could lead to a device isolation problem, whereas the limited capacity of the controller machine including computing power and storage resources could lead to a controller overload problem [1, 2]. These problems would cause disastrous consequences to the network.

In order to improve the reliability and stability of the network, to architect the control plane with distributed multiple controllers has been investigated [1–3]. Several challenges need to be addressed considering the multi-controller deployment in SDN [4–13]. To be specific, during

✉ Chung-An Shen
cashen@mail.ntust.edu.tw

Yao-ying Tzeng
M10502123@mail.ntust.edu.tw

1 Department of Electronic and Computer Engineering, National Taiwan University of Science and Technology, No. 43, Keelung Rd., Sec. 4, Da'an District, Taipei City 10607, Taiwan, People's Republic of China

the network planning, we need to decide on the number of controllers to be deployed in the network and the location that the controllers reside [7, 8]. Furthermore, we also need to determine which forwarding devices (SDN switches) are managed by which controller. These decisions are made based on the factors such as the capacity of the controller (i.e. how many devices that can be handled by a controller), the reliability of the connection between the controller and the forwarding devices, and the transmission latency between the devices to the controller. Moreover, after the network planning, a management mechanism is required to maintain the reliability of the control plane during execution time [14–18]. In particular, this mechanism monitors the status of the controller and address the scenario of controller failure either due to the disconnection between the controller and the forwarding devices or the controller overload. For example, the approaches reported in [14, 15] are based on the backup controller list where the switch that cannot connect to the controller is directed to other controllers based on the pre-installed list. On the other hand, methods based on the migration algorithms are employed in [16–18] where the switch that cannot connect to the controller is dynamically migrated to another operating controllers. It is mentioned in [26] that the switch migration is required to move from one controller to another when the controller fails. This is an important issue for applications such as Internet of Things (IoT) [4]. The OpenFlow switch specifications [19] also state that the functionality of switch migration is added in the switch so that the new controller can be identified when is necessary.

With the aim of enhancing the reliability and stability of the network, this paper presents an integrated framework for a comprehensive multi-controller management in Software defined networking (SDN). To be specific, the proposed framework is comprised of two phases including the Network Planning phase and the Runtime Maintenance phase. The Network Planning phase contains three stages, Controller Quantity Estimation, Controller Placement, and Switch Assignment. The Controller Quantity Estimation stage estimates the required number of controllers by considering the capacity of the controller and the maximal throughput of the network. Furthermore, the estimated number of controller is used to determine the placement of the controller in the Controller Placement stage by using the proposed Maximal Neighbors Controller Placement (MNCP) algorithm. This algorithm intends to identify the node containing the most one-hop neighbors as the controller node so that the problem of device isolation for the controller is minimized. Finally, in the Switch Assignment stage, a Loading-based Hop Count Switch Assignment (LHCA) algorithm is proposed to assign each switch to its managing controller. In addition, the Runtime Maintenance phase aims to maintain the reliability and stability of the control plane during the execution time. In particular, a mechanism based on the State Behavior Tree (SBT) is proposed in this phase to manage the loading of the controller and to handle the scenario of the controller overload. This SBT-based mechanism migrates the switches that are managed by the controller with 90% loading to another controller.

In summary, the main contributions of this paper can be considered as follows.

- A comprehensive and integrated framework of the multi-controller management scheme is presented in this paper. This framework includes a Network Planning phase for the controller deployment and a Runtime Maintenance phase to maintain the reliability of control plane during the execution time.
- The MNCP algorithm and the LHCA algorithm are proposed in the Network Planning phase for the controller placement and switch assignment. These two algorithms are designed with the aiming of mitigating the device isolation problem and to increase the device connectivity between the controller to its managed switches. The experimental results show that, compared to the prior arts, the proposed algorithms reduce the isolation probability up to 89% and increase the device connectivity by up to 34%.
- A mechanism based on the State Behavior Tree (SBT) is proposed in the Runtime Maintenance phase approach based on the is proposed in this paper. This mechanism dynamically manages the loading of the controller by migrating the switches that are managed by the controller with 90% loading to another controller. The experimental results show that, compared to the prior arts, the proposed mechanism significantly decreases the occurrence of the controller overload.

The rest of this paper is organized as follows. Section 2 discusses the background and the related work. Section 3 describes the overview of proposed framework. Sections 4 and 5 introduce the detail concept and algorithm of the Network Planning and Runtime Maintain stage. The simulation is evaluated in Sect. 6 and the conclusion is in Section 7.

2 Background and related work

The centralized controller in Software-defined networking (SDN) could lead to a Single Point of Failure (SPOF) problem [1, 2] due to the controller overload or device isolation. The SPOF problem is illustrated in Fig. 1 where the controller overload is shown in Fig. 1a and the device isolation is shown in Fig. 1b. The controller could be crashed by the excessive amounts of requests from forwarding devices, e.g. SDN switches. The controller could also be disconnected

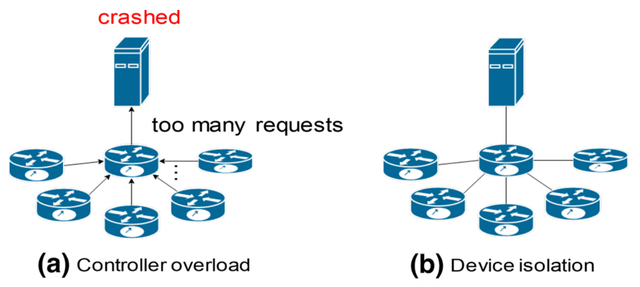

Fig. 1 The illustration of Single Point of Failure (SPOF) problem where (**a**) is the controller overload and (**b**) is the device isolation

from the switches due to the malfunction of the controller or disconnection of the network. The SPOF problem of the controller heavily degrades the stability and reliability of the network. Such problem becomes especially important for the network environment like factory, smart city, smart grid, and IoT [1–4]. For example, the disconnections with the controller could result in the loss of critical information in the IoT environment or lead to disruption of production in the factory. In order to enhance the reliability and stability of the network, it is proposed to architect the control plane by deploying multiple controllers in SDN [1–3]. In this regard, a multi-controller management scheme is required to make design decisions such as the number of controllers that are required to be deployed and the placement of the controller in the network [4, 5]. Furthermore, how to assign the managing controller for each forwarding device in the network is decided [6–8].

Approaches have been proposed in literature aiming to design the multi-controller management scheme for a highly reliable and stable SDN. In particular, the approach proposed in [2] determines the required number and the placement of the controller by considering the path diversity. However, the approach proposed in [2] could increase the loading of the controller. The work presented in [5] considers the link failure of the network and the methodology proposed in [6] estimates the required number of the controller by considering the latency of the network. Furthermore, an evolutionary algorithm is presented in [7] to maximize the connections between the control plane and the data plane and to balance the loading of each controller. Moreover, a well-known K-Means algorithm is shown in [8] to decompose the network into clusters containing the lowest distance from the switches to their managing controllers. In addition, the work in [10] conducts the controller placement by minimizing the connectivity loss, the work in [12] considers the latency of the network and the capacity of the controller, and the work in [13] places the controller by minimizing the worst-case latency from the switch to controller. However, these approaches do not mention how to efficiently identify backup controller when the failure occurs.

On the other hand, the work in [2] and [14–16] proposes to pre-compute and pre-install a backup controller list in each switch. Once the connection between the switch and its managing controller is lost, another controller in the backup controller list is assigned as the substitute controller. Furthermore, a failover mechanism to migrate the switches from the failed controller to the backup controller is presented in [14] and [15], whereas a model has been presented in [16] for collecting the information about the network and for migrating the switches of the overloading controller to the backup controller. Furthermore, the work presented in [17] is about the flow-based dynamic management scheme so that the loading of the controller is monitored in real-time and the migration of the switches are conducted for balancing the loading of the controller.

However, to the best of our knowledge, a comprehensive multi-controller management scheme including network planning and runtime maintenance has not been reported yet in the literature. This paper presents an integrated framework for a comprehensive multi-controller management scheme in Software defined networking (SDN). The proposed framework is comprised of a Network Planning phase and a Runtime Maintenance phase. In the Network Planning phase, we propose approaches to estimate the required number of controllers in the network, to identify the locations for accommodating the controllers. We also propose an approach for dynamic allocation of switches to those controllers. Furthermore, the proposed approaches are designed so that the occurrence of the controller overload and the device isolation can be minimized. Moreover, in the Runtime Maintenance phase, a State Behavior Tree (SBT) based algorithm is proposed to manage the load balancing of the controllers. In particular, once the loading of a controller exceeds 90% of its capacity, the switches that is handled by that controller is migrated to be handled by other controllers. As a result, the problem of controller overload is greatly mitigated.

3 The proposed framework for multi-controller

3.1 The overview of the proposed framework

Figure 2 presents the proposed integrated framework for the comprehensive multi-controller management in SDN. It can be seen in Fig. 2 that this framework is comprised of two phases namely Network Planning and Runtime Maintenance. To be specific, the Network Planning phase considers the deployment of multiple controllers and contains three functional blocks of Controller Quantity Estimation, Controller Placement, and the Switch Assignment. These functional blocks are performed to

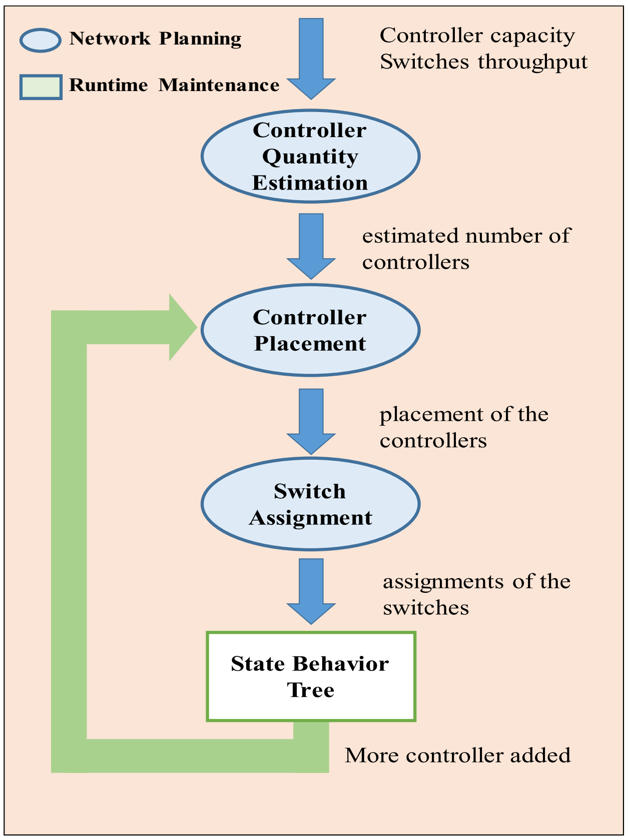

Fig. 2 The overview of the proposed integrated framework of multi-controller management scheme

estimate the number of required controllers in the network, to determine the node for placing controllers in the network, and to assign each switch to its managing controller. Furthermore, the primary goal of the Network Planning phase is to enhance the reliability and stability of the control plane by minimizing the occurrence of the device isolation and controller overload.

Moreover, in order to maintain the reliability and stability of the control plane after the multi-controller deployment, the Runtime Maintenance phase is proposed to manage the control plane of the network during the execution time. An algorithm based on the State Behavior Tree (SBT) is proposed in the Runtime Maintenance phase to operate the corresponding reactions according to the situation of the controllers. In particular, once the loading of a controller exceeds 90% of its capacity, the switches that is handled by that controller is migrated to be handled by other controllers. As a result, the problem of controller overload is greatly mitigated. In the following, the approaches employed in the proposed framework are discussed in details.

Table 1 Parameters used in the Controller Quantity Estimation

Q	The estimated number of controllers
$Capacity(c)$	Responses of packet-in message per millisecond [20]
$Throughput(sw)$	Maximum packet-in message per second [21]
T_{sym}	The synchronization cost of controller [16]
d_i	The size of synchronize data of controller i
t_{ij}	The transmission time to send data from controller i to controller j

3.2 The network planning phase

It can be observed from Fig. 2 that three tasks need to be performed during the Network Planning phase of the proposed framework, including the estimation of the required number of the controller in the network (i.e. the Controller Quantity Estimation shown in Fig. 2), the determination of the node to accommodate the controller (i.e. the Controller Placement), and assignment of the switches to the managing controller (i.e. the Switch assignment). Moreover, in the proposed framework, these functions are performed with the aim of enhancing the reliability and stability of the control plane. In particular, the Controller Quantity Estimation estimates the required number of controllers in the network by considering the capacity of the controller as well as the throughput from the switches. Furthermore, the Controller Placement determines the location of the controller by maximizing the connections between the controllers and switches, whereas the Switch Assignment assigns the switches to the managing controllers by jointly considering the loading of the controller and the connections.

3.2.1 Controller quantity estimation

In the proposed framework, the functional block of Controller Quantity Estimation is employed to estimate the number of controllers to be deployed in the network. Furthermore, in order to minimize the occurrence of the controller overload and device isolation for the control plane, the Controller Quantity Estimation takes multiple network constraints and information into consideration such as the capacity of the controller, the overall throughput of the network, the requests from the switches to the controllers, the inter-controller synchronize cost [16], and the possible packet-in messages of each switch. Specifically, the number of deployed controllers in the network is determined by satisfying the constraint expressed in Eq. (1)

$$\sum_{i=1}^{n} Throughput(sw_i) \le \sum_{k=1}^{Q} Capacity(c_k) \tag{1}$$

where the parameters for the estimation is summarized in Table 1. It is noted that in this constraint we assume a worst scenario situation where the switches generate the packet-in message of each new packet.

The fundamental concept expressed in Eq. (1) is that the accumulated loading of switches in the entire network should be less than or equal to the overall capacity of all controllers in the network. However, in practice, the increased number of controllers in the network incur severe overhead of synchronization cost between the controllers. As a result, the proposed estimation approach further takes the synchronization cost into the consideration. It is expressed in Eq. (2) that the estimated Q minimizes the summation of maximal inter-controller synchronization cost noted as T_{sym}.

$$\min T_{sym} = \sum_{i,j \in Q, i \neq j} d_i \cdot t_{ij} \tag{2}$$

Furthermore, the parameter d_i presents data d of controller i that synchronizes with other controllers, and the t_{ij} is the required maximal round trip time for data transmission from controller i to controller j.

3.2.2 Controller placement

After the number of controllers to be deployed Q is estimated, the placement of the controllers is determined by the Controller Placement functional block. In order to mitigate the SPOF problem by minimizing the device isolation, a Maximal Neighbors Controller Placement (MNCP) algorithm is proposed in our framework. In particular, with the aim of increasing the potential connections between the control plane and data plane, the MNCP algorithm intends to select the nodes to accommodate the controller such that the connections between the switches and the controllers are maximized. The flowchart of the proposed MNCP algorithm is presented in Fig. 3 and the parameters used in this algorithm is summarized in Table 2. It is shown in Fig. 3 that this algorithm is comprised of two inputs including the number of controller Q estimated from the Controller Quantity Estimation and the undirected graph of the network G. Furthermore, the graph is represented as $G=(V, E)$ where V denotes the node in the network and E is the edge in the network. The first step of MNCP is to calculate the number of neighbors of each node that is represented by nb (v_i) given the node v_i from the graph G. In the following, the maximal number of those neighbors, noted as Maxnb, is used as the selection criteria to identify the candidate controller nodes Cs through the execution of the allv function.

In the final stage of MNCP, it is checked that if the sum of the number of nodes in the controller set C and the candidate set Cs is equal to the required number of controller Q. If the selected controller node is smaller than Q, this algorithm will put the nodes in Cs into C and set a lower neighbor number to identify a new Cs until the check is satisfied. On the other hand, if the number of selected controller node is larger than Q, the Q nodes containing the largest hops from the nodes in C set is selected by performing the mhc function. The set of Cs is added into the set of C, and if the number of the nodes in C is equal to Q, C will be the result of the Controller Placement. In other words, the resulted set of C for the Controller Placement always limited by the constraint expressed Eq. (3) where the x_{ij} represents the connection from controller i to switch j and if there is a direct connection from i to j, $x_{ij} = 1$, otherwise $x_{ij} = 0$.

$$\max \sum_{i \in C, j \in S, C, S \in G} x_{ij}, \text{where} x_{ij} = \{0, 1\} \tag{3}$$

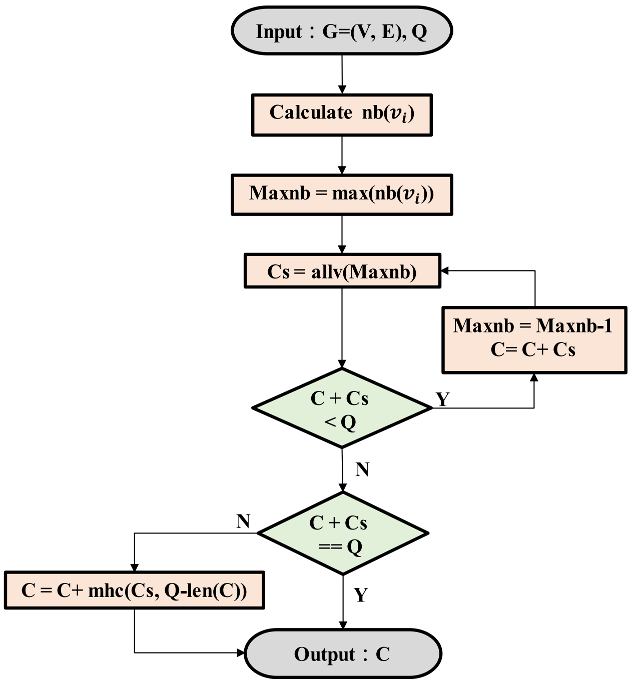

Fig. 3 The flowchart for the proposed Maximal Neighbors Controller Placement (MNCP) algorithm

Table 2 Parameters used in the proposed MNCP algorithm

Q	The required controller quantity
nb(v_i)	The number of neighbor node of vertex v_i. $\forall$ $v_i \in V$
Cs	The candidate node of controller
C	The set of controller nodes
allv(x)	The vertexes which has neighbor value x
mhc(Cs, m)	Find the required number of nodes m which has the max hop with known controllers

3.2.3 Switch assignment

The proposed MNCP algorithm delivers the results of controller placement by mitigating the problem of device isolation. At the final stage of the Network Planning phase, a Loading-based Hop Count Assignment (LHCA) algorithm is proposed to assign each switch in the network to its managing controller. In order to mitigate the problem of controller overload, the LHCA algorithm intends to assign the switch to its managing controller with a constraint that the loading of the controller cannot exceed its capacity. Figure 4 presents the flowchart of the proposed LHCA algorithm where the parameters used in this algorithm are summarized in Table 3. It is shown in Fig. 4 that the undirected graph of the network *G* and the set of the controller C are from the MNCP algorithm and are the inputs to the LHCA algorithm. In the LHCA algorithm, the number of least hops from every switch to each controller is calculated by the

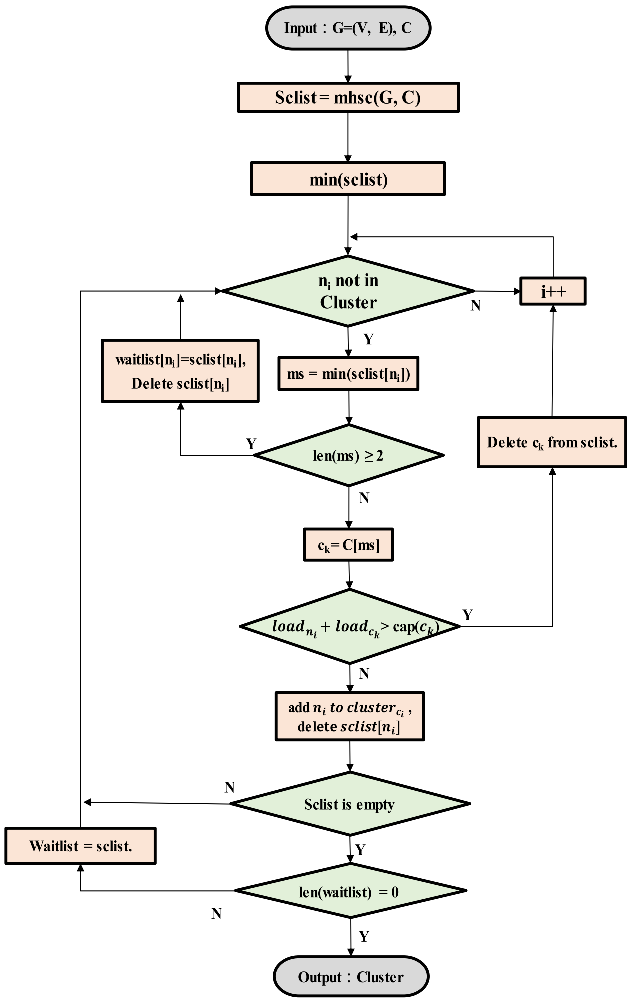

Fig. 4 The flowchart of the proposed Loading-based Hop Count Assignment (LHCA) algorithm for the switch assignment

Table 3 Parameters used in the proposed LHCA algorithm

mhsc(G, C)	The minimum hop count from all the controller to every switches
min(sclist[n_i])	The minimum hop count from n_i to controller
len(x)	The length of list x
$load_{c_k}$	The current load of c_k.

mhsc(G,C) function and is noted as Sclist. In the following, this algorithm identifies the unassigned switch which has the minimum hops with the closest controller and assigns this switch to the nearest controller. If the number of the nearest controllers is more than one, the switch will be stored into a waitlist and is assigned only after the Sclist is empty.

Moreover, when a switch is assigned to a controller, the LHCA algorithm checks if the controller is overloaded. If the controller is overloaded, the controller is removed from the Sclist and the second nearest controller is assigned to the switch instead. Finally, all the switches are assigned and the assignment set is given as Cluster. In particular, the resulted Cluster set of switch assignment is limited by the constraint shown in Eq. (4) where the load(c_i) represents the summation of the maximal switch throughput which is managed by controller c_i. The constraint expressed in Eq. (4) also represents that the loading ofc_i, load(c_i), must not exceed the controller capacity of c_i in C set.

$$Capacity\left(c_i\right) > max\ load\left(c_i\right), c_i \in C \quad (4)$$

where load(c_i) = $\sum_{j=1}^{n} S_j$, ∀j∈ *Clusteri*.

It is noted that the switch with less hop counts and the loading of the controller are both taken into the consideration. In other words, if there are two or more switches with the same hop count, the one that is detected first will be taken for evaluating the loading of the controller.

3.2.4 The runtime maintenance phase

After the deployment of control plane is determined through the Network Planning phase of the proposed framework, the Runtime Maintenance phase is executed to manage the reliability and stability of the control plane based on the multi-controller scheme. According to the requirements of the network, a typical runtime management considers the failure precaution, the reaction of the control plane if the controller failure occurs, the load balance of the control plane, and the latency between the control plane and the data plane. To be specific, the Runtime Maintenance mechanism proposed in this framework focuses on the load balance of the control plane and the restoration scheme for the scenario of controller failure.

An approach based on the State Behavior Tree (SBT) is proposed in the framework to manage the restoration mechanism after the controller failure. The structure of the proposed State Behavior Tree (SBT) is shown in Fig. 5 includes four main blocks namely Priority Setting, Reserved, Other controller failure, and Restore. Each of these four blocks is corresponding to a scenario that is taken care of during the Runtime Maintenance phase. Each controller in the network will be given a priority and the scenario regarding to each controller will be monitored by the SBT. The decision corresponding to each controller like migration or switch reassignment will be determined. It is noted that the proposed mechanism assumes that there are keep-alive messages between the controllers communicating the loading and the synchronize data of the controllers.

The Priority setting block shown in Fig. 5 sets the priority for each controller according to the loading of the local controller. The priority setting serves as the basis to determine whether the controller has margin of capacity to receive additional switches from other controllers. This block uses a priority flag to reflect the status of loading for any given controller. Specifically, the priority flag for any local controller is set to 1 when the loading of that controller is less than 50% of its overall capacity. On the other hand, the priority flag is set to 0 when the loading of the controller is large than or equal to 50% of its overall capacity. Furthermore, the Reserved block shown in Fig. 5 manages the scenario of controller overload. Specifically, when the loading of a local controller exceeds 90 percent of its overall capacity, it will attempts to identify another target controller and starts to migrate the switches to that newly identified target controller until the loading of the local controller is below 90 percent. If no controllers can be identified, the local controller goes to the neighboring controller with the minimal loading as the target controller and migrates the switch only if the loading for the target controller is less than the local controller. If the loading for all the neighboring controllers are larger than the local controller, the local controller simply exits the Reserved block.

Moreover, the Other controller failure block shown in Fig. 5 aims to detect the failure of other controllers, i.e., the controllers other than the local controller, through the examination of the synchronization messages. In particular, if the synchronization message from a certain controller is not detected, another controller which contains the lowest loading starts to identifies the switches of that controller and reassigns them by following a reassign procedure shown in Fig. 6. It is sown in Fig. 6 that the controller failing to send out synchronization message Fc, the set of all the controllers C, and the list containing the hop count from each controller to every switch cslist are the inputs to the re-assign procedure. The alive controller sorts the controllers in the set C and identifies the target controller ra_0 containing the minimal loading. The switch which has the minimal hop count in cslist [Fc][ra_0] is assigned and the migration message is sent to the target controller. Afterwards, the controllers are sorted again according to the loading after the re-assignment of the switch. These steps are repeated until all the switches of the failing controller are reassigned. In addition, the Restored block shown in Fig. 5 handles the scenario

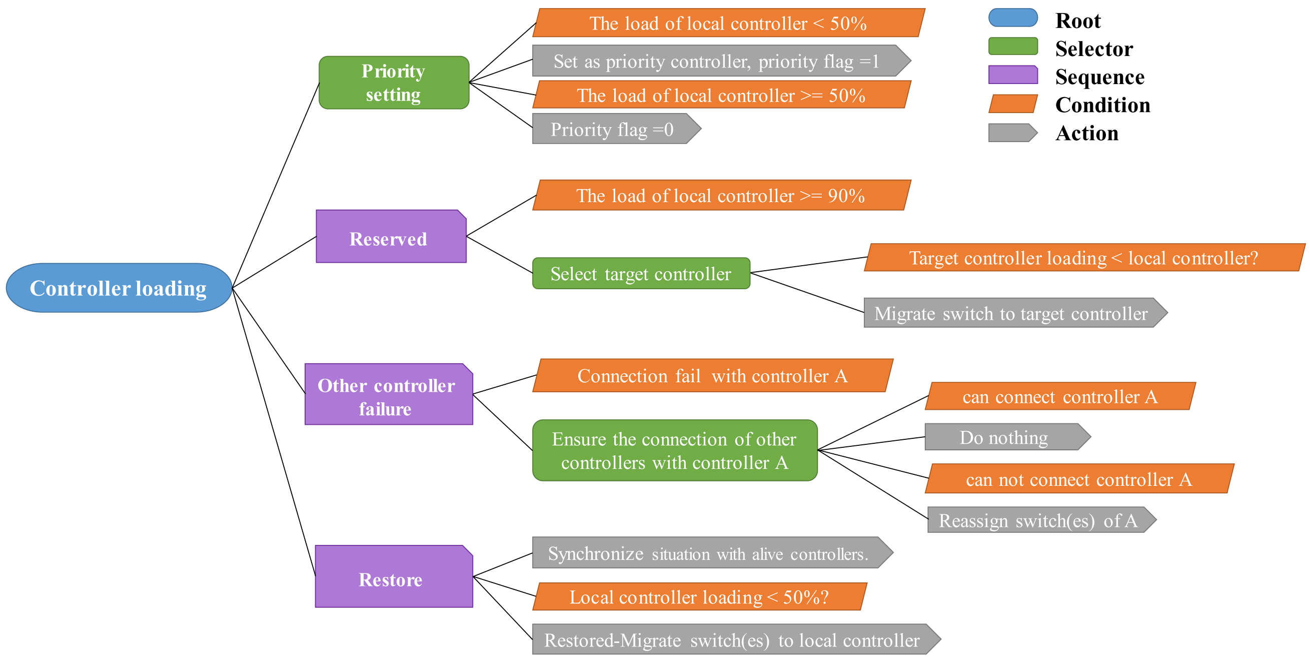
Root
Selector
Sequence
Condition
Action
Controller loading
Priority setting
The load of local controller < 50%
Set as priority controller, priority flag =1
The load of local controller >= 50%
Priority flag =0
Reserved
The load of local controller >= 90%
Select target controller
Target controller loading < local controller?
Migrate switch to target controller
Other controller failure
Connection fail with controller A
Ensure the connection of other controllers with controller A
can connect controller A
Do nothing
can not connect controller A
Reassign switch(es) of A
Restore
Synchronize situation with alive controllers.
Local controller loading < 50%?
Restored-Migrate switch(es) to local controller

Fig. 5 The structure of the proposed State Behavior Tree (SBT) based approach for the Runtime Maintenance phase

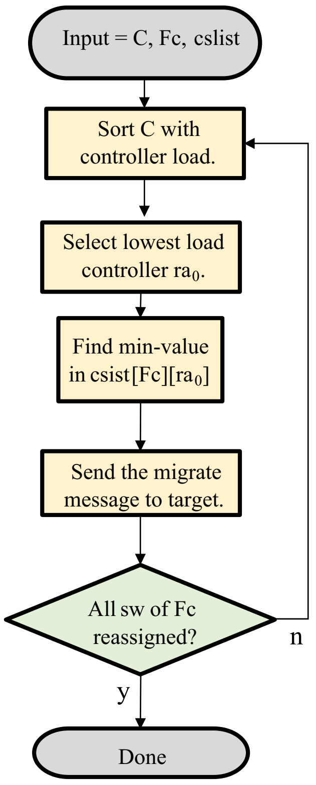
Input = C, Fc, cslist
Sort C with controller load.
Select lowest load controller ra$_0$.
Find min-value in csist[Fc][ra$_0$]
Send the migrate message to target.
All sw of Fc reassigned?
n
y
Done

Fig. 6 The flowchart of the reassign procedure used in the proposed framework

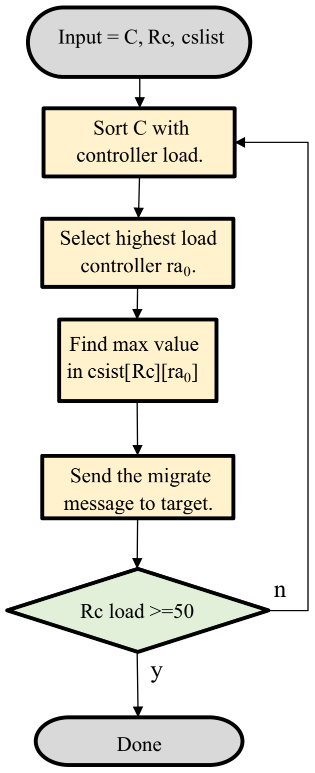
Input = C, Rc, cslist
Sort C with controller load.
Select highest load controller ra$_0$.
Find max value in csist[Rc][ra$_0$]
Send the migrate message to target.
Rc load >=50
n
y
Done

Fig. 7 The flowchart of the restored-migrate procedure used in the proposed framework

where the previously failed local controller is restored from the failure condition. In this case, the restored controller starts to synchronize again with other alive controllers. In order to release the loading of other controllers, the restored controller performs a restored-migrate procedure shown in Fig. 7 to migrate the switches from other controllers to the restored controller. It is shown in Fig. 7 that the restored controller Rc, the set of all the controllers C, and the cslist are the inputs to the procedure. The restored controller sorts the controllers in C and identifies the controller with the maximal loading ra$_{0.}$ The switch that is managed by ra$_0$ and contains the minimal hop count to the restored controller in cslist [Rc][ra$_0$] is migrated to the restored controller. The restore-migrate procedure stops if the loading of the restored controller exceeds 50% of the capacity.

4 Evaluation of the proposed framework

Extensive experiments have been conducted to evaluate the performance of the proposed framework including the Network Planning phase and the Runtime Maintenance phase. The evaluation results are also compared with previously reported approaches. Specifically, the proposed Network Planning approach is compared with the evolutionary algorithm [7] and the K-Means algorithm [8] and the Runtime Maintenance method is compared with the Survivor approach [2] which gives a backup controller list for managing the failure of the control plane. The topologies used for evaluations and comparisons are from the topology zoo [22] and the Internet 2 OS3E topology in the Internet 2 network [23]. All the algorithms are implemented with the python compiler in Ubuntu 14.04 on Virtualbox and executed on the compuer equipped with Intel core i5-4460 processor.

4.1 The evaluation of networking planning

The approaches in the Network Planning phase of the proposed framework are simulated based on five different topologies including the Claranet (15 nodes), the Agis (25 nodes), the Internet 2 OS3E (34 nodes), the Bellcanada (48 nodes) and the Iris (51 nodes). Furthermore, the capacity of the controller is set to be 1800 kilo-requests per second and the maximal throughput of the switch is set to be 200 kilo-requests per second [2]. Moreover, the metrics of isolation probability defined in [24] and device connectivity defined in [25] are used to evaluate the performance of the proposed algorithm and to compare the results with the previously reported approaches. The isolation probability represents the probability that the connections between the controller and its managed switches are all lost [24] where each link is assumed to have the same probability of failure. The detailed definition of the isolation probability is referred to [24]. In addition, the device connectivity [25] represents the overall connectivity from the switches to the so that not only the probability of controller isolation but the average connectivity is considered. The detailed definition of the device connectivity is referred to [25].

The evaluation results for the isolation probability and device connectivity based on the five experimented topologies are illustrated in Figs. 8 and 9 respectively. Furthermore, the comparisons between the proposed Network Planning approaches with the evolutionary algorithm proposed in [7] and K-Means algorithm proposed in [8] are also presented in Figs. 8 and 9. These two algorithms are simulated by using the same number of controllers estimated by the Controller Quantity Estimation of the proposed framework. It is shown in Fig. 8 that the average isolation probability of the proposed approach is less than 10% over all the five tested topologies. Compared to the evolutionary algorithm proposed in [7], the isolation probability of the proposed approach is reduced by approximately 67% to 89%. Similarly, compared to the K-Means algorithm proposed in [8], the isolation probability of the proposed approach is reduced by approximately 69% to 89%. In addition, it can be observed from Fig. 9 that the proposed approach increases the device connectivity by 13% to 45% comparing with the evolutionary algorithm [7] and by 13% to 34% comparing with the K-Means algorithm [8]. Therefore, it can be verified from the experimental results that the proposed Network Planning scheme leads to the deployment of multiple controllers with very lower probability of being isolated. Furthermore, more diverse links between the control plane and data plane are resulted. As a result, the reliability and stability of the network are greatly enhanced.

4.2 The evaluation of runtime maintenance

The proposed SBT-based Runtime Maintenance is simulated based on the topology of Internet 2 OS3E (34 nodes)

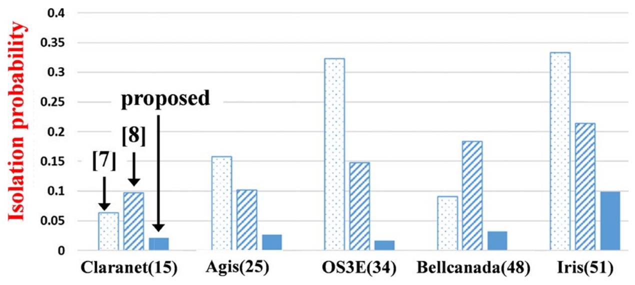

Fig. 8 The evaluation results and comparison with prior arts for the isolation probability

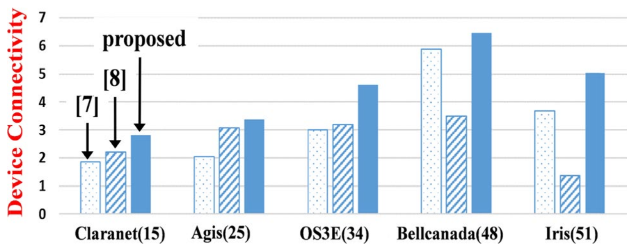

Fig. 9 The evaluation results and comparison with prior arts for the device connectivity

[23] and is compared with the approach proposed in [2] that uses residual capacity-based heuristic backup controller list for handling the controllers failure. The approach in [2] migrates the switches to another controller containing sufficient capacity in the backup list. Furthermore, both of the proposed algorithm and the algorithm in [2] are on the basis of the results of Network Planning including the controller placement and the switch assignment. Two experiments are conducted to analyzing the loading of the controllers and examining the number of times that the controller is overloaded.

In order to analyze the loading of the controllers, the initial throughput of the switch is set to be 150 kilo-requests per second and the throughput of the switch that is managed by the tested controller increases by 1 kilo-request per second once at a time until one of the examined controllers is failed. The experiment results for the loading of controllers versus the added throughput to the switch are summarized in Figs. 10, 11, 12, 13 where each figure is comprised of four sub-plots showing the loading of the controller that is place on node 3, node 24, node 28, and node 32 respectively.

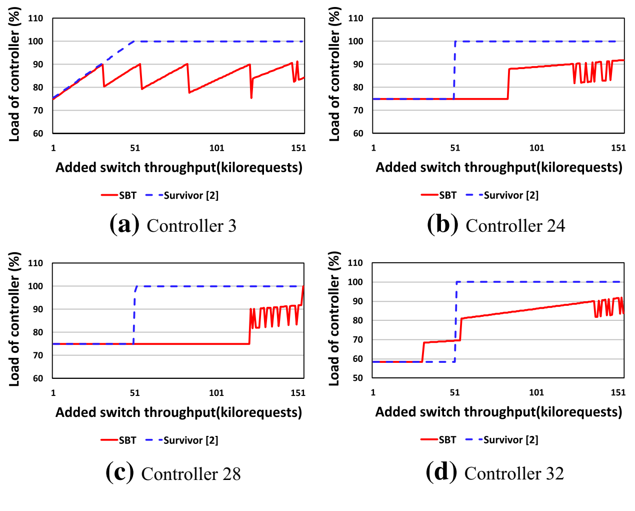

Fig. 10 The loading of each controller (**a–d**) for the proposed approach and the approach in [2] when the throughput of switches managed by controller 3 increases

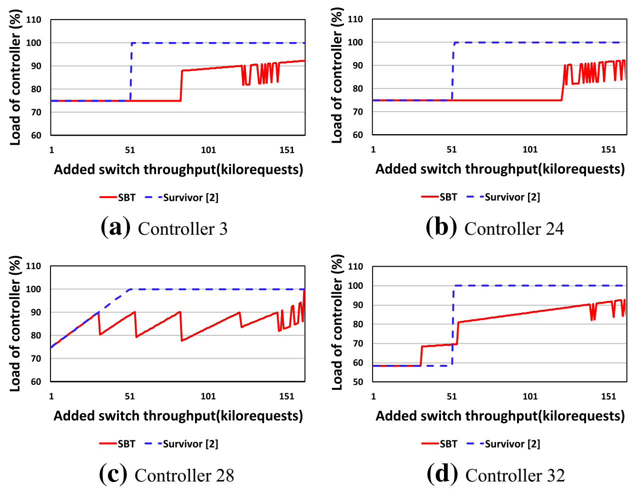

Fig. 11 The loading of each controller (**a–d**) for the proposed approach and the approach in [2] when the throughput of switches managed by controller 28 increases

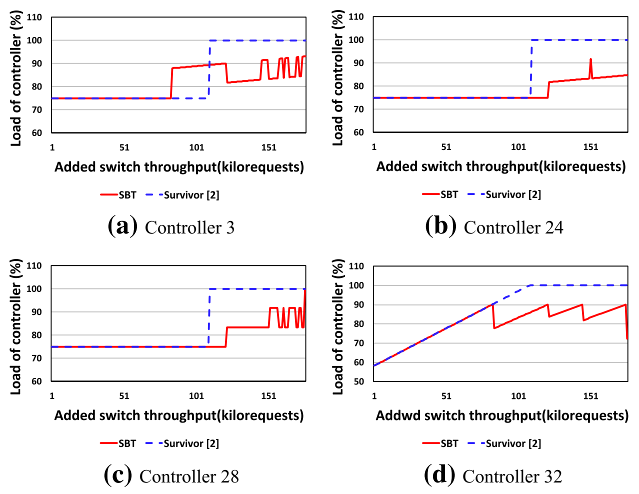

Fig. 13 The loading of each controller (**a–d**) for the proposed approach and the approach in [2] when the throughput of switches managed by controller 32 increases

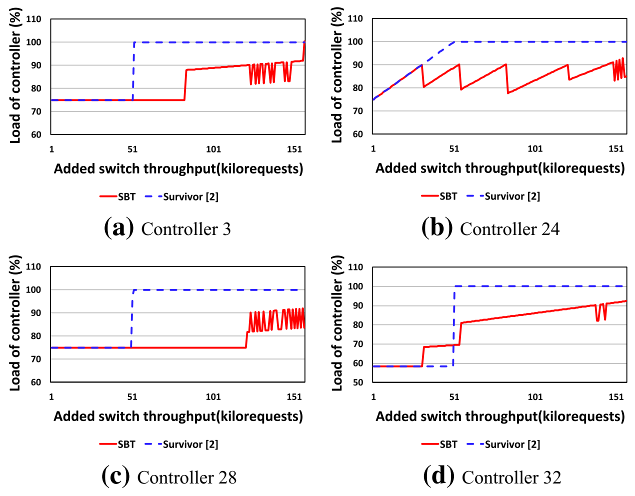

Fig. 12 The loading of each controller (**a–d**) for the proposed approach and the approach in [2] when the throughput of switches managed by controller 24 increases

Furthermore, each figure illustrates a case that the throughput of the switches belongs to a controller increases. For example, the case where the throughput of the switches belongs to controller 3 is shown in Fig. 10, the case for the controller 28 is shown in Fig. 11, the case for the controller 24 is shown in Fig. 12, and the case for the controller 32 is shown in Fig. 13. Moreover, the simulation results based on the proposed SBT as well as the algorithm of [2] are illustrated.

It is shown in Fig. 10a that the loading of the controllers for the proposed approach and [2] are closely overlapped until the loading of controller 3 reaches 90%. At the point where the loading of controller 3 reaches 90% the first time, the proposed approach keeps reducing the loading of controller 3 by migrating the switches of controller 3 to other controllers. While other controllers also reach the loading of 90%, the Reserved block in the proposed SBT structure is conducted to reduce the loading. On the other hand, for the approach of [2], the loading of controller 3 keeps increasing after reaching 90%. The controller overload occurs when the throughput of selected switches is 200 kilo-requests per second. Thus, it can be observed from Fig. 10 that through the proposed SBT algorithm, the loading for all four examined controllers are maintained below 90% mostly and the occurrence of the controller overload is significantly delayed. The trend similar to Fig. 10 can also be observed in Figs. 11, 12, and 13. The controller that contains the throughput-increased switches actively migrates its managed switches to other controllers for reducing the loading. As a result, compared to the approach used in [2], the occurrence of the controller overload is greatly delayed for the proposed SBT approach.

In addition, in the second experiment the throughput of switches which belong to the selected controller is set from 150 kilo-requests per second and added 1 kilo-requests per second once a time until the summation of the throughput in the network is equal to the summation of the capacity of the deployed controllers. The controller is always set to restore after the failure occurs. Figure 14 shows the maximal failure controllers at the same time of four different cases as the throughput-increased switches belong to controller 3, 24, 28 and 32 initially. It can be observed from Fig. 14 that the maximal failure controllers at the same time for the proposed algorithm is smaller than the approach in [2] for all four cases. Therefore, based on the illustrated experimental results, it can be verified that the proposed multi-controller

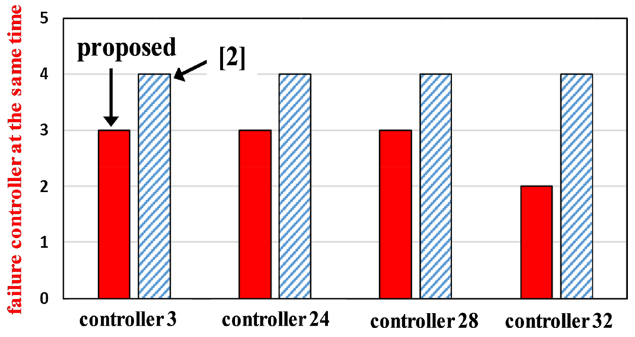

Fig. 14 The maximum failure controller at the same time in the network

management framework greatly enhances the reliability and stability of the network.

5 Conclusion

An integrated framework for a comprehensive multi-controller management in SDN is presented in this paper. The proposed framework is comprised of a network planning phase and a runtime maintenance phase. The algorithms proposed in the network planning phase estimates the require number of the controllers in the network, determines the node for placing the controller, and assigns the switch to its managing controller. The problems of device isolation and controller overload are mitigated such that the reliability and stability of the control plane can be enhanced. Moreover, a mechanism based on the State Behavior Tree is proposed during the runtime maintenance phase of the framework. This mechanism manages the loading of the controller during the execution time so that the occurrence of the controller overload is minimized. The experimental results have shown that the proposed framework reduces the isolation probability by up to 89% and increases the device connectivity by up to 34%. Moreover, the occurrence of the controller overload during runtime can be significantly decreased.

Funding This work is supported by Ministry of Science and Technology, Taiwan.

References

1. Ahmad, S., & Mir, A. H. (2021). Scalability, consistency, reliability and security in sdn controllers: A survey of diverse SDN controllers. *Journal of Network and Systems Management, 29*(1), 1–59.
2. Müller, L. F., Oliveira, R. R., Luizelli, M. C., Gaspary, L. P., & Barcellos, M. P. (2014) Survivor: An enhanced controller placement strategy for improving SDN survivability. In *2014 IEEE Global Communications Conference* (pp. 1909–1915).
3. Hassas Yeganeh, S., & Ganjali, Y. (2012). Kandoo: a framework for efficient and scalable offloading of control applications. In *Proc. SIGCOMM HotSDN workshop* (pp. 19–24). ACM, 2012.
4. Bekri, W., Jmal, R., & Chaari Fourati, L. (2020). Internet of things management based on software defined networking: A survey. *Springer Journal of Wireless Information Networks, 27,* 385–410.
5. Zhong, Q., Wang, Y., Li, W., & Qiu, X. (2016). A min-cover based controller placement approach to build reliable control network in SDN. In *NOMS 2016—2016 IEEE/IFIP Network Operations and Management Symposium*, Istanbul, 2016 (pp. 481–487).
6. Jiménez, Y., Cervelló-Pastor, C., & García, A. J. (2014) On the controller placement for designing a distributed SDN control layer. In: *2014 IFIP Networking Conference*, Trondheim, 2014 (pp. 1–9)
7. Sanner, J. M., Hadjadj-Aoul, Y., Ouzzif, M. & Rubino, G. (2017). An evolutionary controllers' placement algorithm for reliable SDN networks. In *2017 13th International Conference on Network and Service Management* (*CNSM*), Tokyo, 2017 (pp. 1–6).
8. Wang, G., Zhao, Y., Huang, J., Duan, Q., & Li, J. (2016). A K-means-based network partition algorithm for controller placement in software defined network. In *2016 IEEE International Conference on Communications (ICC)*, Kuala Lumpur, 2016 (pp. 1–6).
9. Mendiola, A., et al. (2019). Enhancing network resources utilization and resiliency in multi-domain bandwidth on demand service provisioning using SDN. *Springer Journal of Telecommunication Systems, 29*(1), 505–515.
10. Zhang, Y., Beheshti, N., & Tatipamula, M. (2011) On resilience of split-architecture networks. In *2011 IEEE Global Communications Conference—GLOBECOM 2011*, 2011 (pp. 1–6).
11. Mohammadi, R., Javidan, R., Keshtgari, M., & Akbari, R. (2018). A novel multicast traffic engineering technique in SDN using TLBO algorithm. *Telecommunication Systems, 68*(3), 583–592.
12. Tanha, M., Sajjadi, D., Ruby, R., & Pan, J. (2018). Capacity-aware and delay-guaranteed resilient controller placement for software-defined WANs. *IEEE Transactions on Network and Service Management, 15*(3), 991–1005.
13. Killi, B. P. R., & Rao, S. V. (2018) Link failure aware capacitated controller placement in software defined networks. In *2018 International Conference on Information Networking (ICOIN)*, Chiang Mai, 2018 (pp. 292–297).
14. Li, J., Wang, Y., Li, W., & Qiu, X. (2017). Sharing data store and backup controllers for resilient control plane in multi-domain SDN. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Lisbon (pp. 476-482)
15. Zhang, L., Wang, Y., Li, W., Qiu, X., & Zhong, Q. (2017). A survivability-based backup approach for controllers in multi-controller SDN against failures. In *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Seoul (pp. 100–105)
16. Zhang, J., Hu, T., Zhao, W., & Qiao, D. (2017) DDS: Distributed decision strategy based on switch migration towards SDN control plane. In *2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery* (*CyberC*), Nanjing, 2017 (pp. 486–493).
17. Yao, L., Hong, P., Zhang, W., Li, J. & Ni, D. (2015) Controller placement and flow based dynamic management problem towards SDN. In *2015 IEEE International Conference on Communication Workshop (ICCW)*, London (pp. 363–368)
18. Hegde, S., Ajayghosh, R., Koolagudi, S. G. & Bhattacharya, S. (2017) Dynamic controller placement in edge-core software defined networks. In *TENCON 2017—2017 IEEE Region 10 Conference*, Penang (pp. 3153–3158).
19. ONF. (2015). Openflow switch specification 1.5.1. Available at https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf, 2015.

20. Yao, L., Hong, P. & Zhou, W. (2014). Evaluating the controller capacity in software defined networking. In *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, Shanghai (pp. 1–6).
21. Rahimi, R. et al. (2016) A high-performance OpenFlow software switch. In *2016 IEEE 17th International Conference on High Performance Switching and Routing (HPSR)*, Yokohama (pp. 93–99).
22. Knight, S., Nguyen, H. X., Falkner, N., Bowden, R., & Roughan, M. (2011). The internet topology zoo. *IEEE Journal on Selected Areas in Communications, 29*(9), 1765–1775.
23. Internet2 Open science, scholarship and services exchange. [Online]. Available: http://www.internet2.edu/network/ose/
24. Papoulis, A. (1964). The meaning of probability. *IEEE Transactions on Education, E-7*(2–3), 45–51.
25. Beineke, L. W., Oellermann, O. R., & Pippert, R. E. (2002). The average connectivity of a graph. *Discrete Mathematics, 252*(1), 31–45.
26. Al-Tam, F., & Correia, N. (2019). On load balancing via switch migration in software-defined networking. *IEEE Access, 7,* 95998–96010.