Front. Comput. Sci., 2020, 14(1): 146–161
https://doi.org/10.1007/s11704-019-7266-6

RESEARCH ARTICLE

AAMcon: an adaptively distributed SDN controller in data center networks

Waixi LIU (✉), Yu WANG, Jie ZHANG, Hongjian LIAO, Zhongwei LIANG, Xiaochu LIU

Department of Electronic and Information Engineering, Guangzhou University, Guangzhou 510006, China

Abstract When evaluating the performance of distributed software-defined network (SDN) controller architecture in data center networks, the required number of controllers for a given network topology and their location are major issues of interest. To address these issues, this study proposes the adaptively adjusting and mapping controllers (AAMcon) to design a stateful data plane. We use the complex network community theory to select a key switch to place the controller which is closer to switches it controls in a subnet. A physically distributed but logically centralized controller pool is built based on the network function virtualization (NFV). And then we propose a fast start/overload avoid algorithm to adaptively adjust the number of controllers according to the demand. We performed an analysis for AAMcon to find the optimal distance between the switch and controller. Finally, experiments show the following results. (1) For the number of controllers, AAMcon can greatly follow the demand; for the placement location of controller, controller can respond to the request of switch with the least distance to minimize the delay between the switch and it. (2) For failure tolerance, AAMcon shows good robustness. (3) AAMcon requires less delay to the network with more significant community structure. In fact, there is an inverse relationship between the community modularity and average distance between the switch and controller, i.e., the average delay decreases when the community modularity increases.(4) AAMcon can achieve the load balance between the controllers. (5) Compared to DCP-GK and *k-critical*, AAMcon shows good performance.

Received July 30, 2017; accepted January 30, 2018

E-mail: liuwaixi@gmail.com

Keywords software defined network, controller placement, community, adaptively adjusting

1 Introduction

Software-defined network (SDN) [1,2] separates the centralized control plane from the distributed data plane. With SDN's support of flexible network management and rapid deployment of new functionalities, there is an increasing interest in deploying SDN in both inter-data center (e.g., Google B4 [3]) and intra-data center (e.g., Hedera [4]) scenarios.

While the concept of centralized control is the foundation of SDN, implementing it on a centralized controller does not provide the required levels of availability, responsiveness, and scalability. To improve scalability and avoid a single point of failure,some recent works have explored architectures for building distributed SDN controller plane [5] (e.g., NVP [6]). In generally, switches are statically assigned to one or multiple controllers in these distributed controller planes.

However, static assignment between switches and controllers leads to long and highly varying controller response time, simply because traffic in data center networks (DCN) frequently fluctuates with both temporally and spatially [7,8]. Spatially, switches in different layers of the DCN topology experience significantly different flow arrival rates. Temporally, the aggregate traffic usually peaks in daytime and falls at night. Moreover, traffic variability also exists in shorter time scales even when the total traffic remains unchanged. For example, measuring results over real data centers have shown that the peak-to-median ratio of low arrival rates is almost 1–2 orders of magnitude [9]. All these factors cause hot

spots among some controllers, resulting in excessively long response time for the switches they manage. Although the controller response time may not be significant for elephant flows, it fundamentally limits the network's ability to quickly react to changes such as failures and may cause transient congestion to last for a long time [8].

Therefore, for software defined DCN, using dynamic switch assignment is important to obtain lower controller response time and better utilization of controller resources. Dixit et al. [10] propose an efficient protocol to enable switch migration across multiple controllers without message loss or observable delay. However, how to determine the assignment remains open. On the one hand, from the switch's view, it prefers a controller with low response time to improve performance. On the other hand, from the controller's view, it is more willing to manage topologically closer switches to reduce the control traffic overhead. This is critical when the communication between switches and controllers is frequent and occupies scarce bandwidth resources. These preferences are always intertwined, and make the problem especially challenging. Furthermore, the following challenges also need to be addressed:

1) Centralization places heavy burden on the software-based controller. When a network event occurs (e.g., a host moves from one location to another or a piece of hardware fails), the control flow has to make an indirection via the controller. As a result, the reaction time can be orders of magnitude slower than an in-network reaction [11]. This is particularly problematic in application scenarios that require fast response to frequent network dynamics.

Recently, to the slow reaction problem for important network events, some works have begun to explore a radically different approach—offloading some latency-sensitive network management tasks to the data plane itself (stateful data plane), and implementing the tasks using only the existing rule-based infrastructure already implemented in the switches [12]. This in-network approach is appealing due to two reasons: a) It eliminates the extra round trip to the controller, and b) It uses only a hardware-based implementation of forwarding rules, instead of software-based controller logic, bring a significant performance boost.

In SDN, flows can be configured with proactive or reactive mode. The time associated with the reactive flow setup is the sum of the time it takes to send the packet from the switch to the controller, the processing time in the controller, and the time it takes to send the configuration message back to the switch. Therefore, there are two main factors influencing the establishment time of flow: the distance between switch-controller and the processing time of the controller. For the former, this paper attempts to place the controller closer to the switch based on the stateful data plane (localized local controller).

2) Another key limitation of past works is that the mapping between a switch and controller is statically configured. This results in long and highly varying controller response time-dueto DCN traffic varying with both temporally and spatially as mentioned above.

Since the initial proposal of the network function virtualization (NFV) [13] concept, its relationship with SDN was argued to be complementary and potentially of benefit when both technologies are combined [14].

Thus, this paper attempts to adjust adaptively the number of controllers according to the demand by the NFV.

3) We demonstrate that a good selection of controllers may balance the load among them and also reduce the data loss in the control layer. As a result, the selected controllers can efficiently distribute the management duties among them to improve the scalability of the management process where each controller operates on its own abstract view of the network. This study attempts to make the switch to select the right controller based on the complex network community theory.

To address the abovementioned challenges, this study proposes the adaptively adjusting and mapping controllers (AAMcon). The contributions of this study are as follows:

a) We propose an elastic distributed SDN controller architecture for DCN, where the controller pool based on SDN+NFV dynamically expands or shrinks as the dynamic demand for controller due to the aggregate load changing over time. Where we propose a fast start/overload avoid algorithm to adaptively adjust the number of controllers according to the demand of switch.

b) We propose an efficient switch-to-controller mapping scheme by building a community that contains a local controller on the switch using the complex network community theory. Where the local controller is placed at the most important node in the community and becomes the first mapping choice of the switch when it needs support from the controller. The controllers can respond to the switch with the least distance to minimize the communication time between the controller and switches. Simultaneously, AAMcon can achieve load balancing between controllers, avoid congestion, and is very tolerant to failures.

c) This study proposes a stateful data plane to place the controller closer to the switch.

The remainder of the paper is organized as follows. In Section 2, related works are introduced. In Section 3, the design and implementation for building a scalable control layer are presented. In Section 4, the experiment results of the proposed method are presented. Section 5 discusses some related issues. Finally, conclusion and future work are presented in Section 6.

2 Related work

2.1 Distributed SDN controllers

The SDN controller is responsible to control flows and manage network resources based on the global view of the entire network. However, to maintain a global view, controllers have to synchronize states with each other. As the state of the whole network frequently changes, synchronization may lead to network overload. Moreover, Levin et al. [15] have found that the inconsistent control state of the SDN significantly degrades the performance of many applications. Therefore, the control plane state and logic must be physically distributed. Then, some proposals attempted to construct the local network view.

For OpenFlow rule allocation and endpoint policy enforcement, a linear optimization model in resource-constrained SDN networks with relaxed routing policy is proposed [16]. Where they have shown that the general problem is NP-hard and proposed a polynomial time heuristic (OFFICER), aiming to maximize the amount of carried traffic in under-provisioned networks.

To reduce the communication overhead and delay of the switch-controller and to achieve the balance of traffic, researchers are exploring the stateful data plane and the local controller. Bianchi et al. [17] opened a number of programmable APIs in the data plane, and presented a finite state machine (XFSMs) to help the switch independently make decisions according to their own local information. Moshref et al. [18] further extended the application categories. They used a state machine (FAST) in the switch to track the flow separate from the state transition table and the action table to reduce their size further, where hash table is used to reduce the delay state table.

Schmid and Suomela [19] used *Local algorithms* (a local distributed computing algorithm) to design a distributed control plane, which divides the network into many regions, and a controller that acts as manager. Therefore, locality task and some global tasks, which can be converted into be local (as an approximate optimal solution), can be solved by the local controller.

Vissicchio et al. [20] developed an architecture that achieves both flexibility and robustness by central control over distributed routing. They introduces fake nodes and links into an underlying link-state routing protocol so that routers compute their own forwarding tables based on the augmented topology.

On the other hand, to achieve the optimal allocation of network resources, the network should be able to adapt to the time-space changes of the consumer's demand. Therefore, the researchers explore the dynamic connection between the switch and controller.

When the switch-controller mapping is statically configured, uneven load distribution will be exhibited among the controllers. To solve this problem, Dixit et al. [10] have proposed ElastiCon that can dynamically increase or decrease the number of controllers according to the change of traffic. In addition, ElastiCon divided the application state, and used the relationship between the switch and application state to achieve the local controller. Moreover, Krishnamurthy et al. [21] proposed an elastic controller assignment mechanism by partitioning the application states and exploring the dependency between switches and applications. They took into account the CPU and memory load, and abstracted the problem to multi-dimensional packing problem for optimization. However, the mechanism does not consider the processing time on controllers, which is a major factor in flow setup time.

In summary, when SDN is used to large-scale real network, the local controller and stateful data plane are required to design for scalability and robustness. They should achieve the dynamic connection between the switch and controller and form a dynamic distributed control plane to improve the real-time and fault recovery capabilities. The distributed SDN controller is discussed in detail in [22].

Differencing from the abovementioned references, we use NFV to design a stateful data plane, where the local controller is abstracted as one virtual network functions (VNF) on switch. A heuristic algorithm is proposed to adjust VNF's number and placement. Therefore, the proposed scheme can be scalable to large-scale network.

2.2 Controller placement problem in software-defined networks

From the theoretical perspective, Yao et al. [23] defined a ca-

pacitated controller placement problem (CCPP), considering the load of controllers. Given a network with fixed topology, this study addressed two issues: how many controllers needed; where each controller is placed. Moreover, Sallahi and St-Hilaire [24] proposed a mathematical model for the controller placement problem aiming to minimize the cost of the network. When a set of switches managed by the controller(s) is given, the model simultaneously determines the optimal number, location, and type of controller(s) as well as the interconnections between all network elements. Ros and Kuiz [25] focused on determining how many controllers need to be instantiated, where they are deployed, and which switches are under the control of each of them to achieve high reliability. However, these schemes assume that the traffic load is fixed and can be looked as the preplanning scheme.

From implementation perspective, Jiménez et al. [26] proposed a distributed protocol called SDN resource discovery protocol (SDN-RDP). *k-critical* [27] algorithm aims to find a placement plan of controller to solve the following two problems: how to build a network topology containing the controller and how to manage this network topology. Essentially, each controller discovers a portion of the network topology creating a minimum latency tree rooted at each controller, thus creating the control layer.

In summary, the controller placement problem is essentially the positioning of a limited number of resources within a network to meet various requirements. These requirements range from latency constraints to failure tolerance and load balancing. In most scenarios, at least some of these objectives are competing. Thus, no single best placement is available, and decision makers need to find a balanced trade-off. Stanislav Lange et al. [28] presented POCO, a framework that provides operators with Pareto optimal placements with respect to different performance metrics.

Summarily, the past work regarded the controller placement problem as a preplanning problem for static traffic. Differencing from abovementioned these, we propose a community-based scheme for the controller placement problem. In addition to taking into account the physical connection between two nodes (e.g., whether there is connection between them, the distance between them), real-time traffic behavior between them is also considered.

2.3 Combination of NFV and SDN (NFV+SDN)

NFV and SDN shared a same objective—using commodity servers and switches, avoiding specific hardware-based components provided by vendors [29]. NFV is able to support SDN by providing the infrastructure where the SDN software can be run [14].

Gember-Jacobson et al. [30] designed an open control plane OpenNF for integration with the SDN+NFV architecture where both the network function (NF) and network forwarding statuses can be quickly and accurately migrated between the NF.

Batalle et al. [31] used NFV to implement the routing function under OpenFlow. Moreover, with the network-as-a-service (Naas) model based on SDN + NFV architecture, Wang et al. [32] proposed a greedy routing protocol with energy saving for the data center network, which can meet the multi-resource constraints.

The open networking foundation (ONF) has proposed “A flexible NFV networking solution” [33], outlining the benefits for the NFV deployment of an OpenFlow-enabled SDN approach to deal with the dynamic provisioning of networking services.

Rodriguez-Natal et al. [34]claim that a decentralization of NFV control while maintaining global state improves scalability, can offer better per-flow decisions and simplifies the implementation of VNF.

From the related work, currently there is a clear momentum for exploiting networking innovation in the light of SDN and NFV.

3 Adaptively adjusting and mapping controllers

3.1 Basic architecture

Reference [35] has shown that community-based data center can efficiently achieve optimal resource management, such as capacity allocation, load balancing, energy optimization, and QoS guarantees. While achieve parallel solving of a huge problem (e.g., MapReduce), the assigned virtual machines (VMs) need to exchange data in order to synchronize their work on the given problem, and we refer it as locality tasks. The inter-VMs communication has to be localized for performance boosting purposes. Thus, we need to assign the VMs to one community and, in this way, localize traffic inside the community, which will reduce the communication delay and increase the overall network performance of the data center. Multi-level community formation algorithms allow for creation of super groups of smaller communities with strong intra and inter community information flows.

Thus, in a community-based software defined data center, this paper designs the stateful data plane: data plane retaining

some control functions instead of a completely dumb terminal as shown in traditional SDN architecture, and the controller is placed on a key node closer to the switch. These can reduce the communication overhead and the round-trip delay between the switch and controller. These can also reduce the controller's workload. Furthermore, the controller is actually localized (local controller), and a hierarchical, distributed controller network is ultimately built. In this controller network, locality tasks are completed by the local controller, and global tasks are completed through the local controller by the global controller as shown as Fig. 1. Thus, the local controller is pushed closer to the switch it control, and can respond to switch's request in real-time and more accurately. At the same time, this paper also designs the stateful data plane to adaptively adjust the number of controllers according to the demand. Notations in this paper is shown in Table 1.

Table 1 Notation acronyms list

Notations	Definition
Q	Modularity for community structure
m	The number of edges for network model
n	The number of vertices for network model
$Betw(v)$	The *betweenness* for node v
Local_C	The number of local controller
load_controller	The load of controller
Threshold	The reserved threshold value for controller numbers
T	One round time for Algorithm 1
θ	The threshold ofdetermining whether controller is overload
Record_data	The local controller number when controller is overload

3.1.1 Background theory

Researches have shown that the community structures in a real network is common, which are an important property of complex networks. For example, tightly connected groups of nodes in a social network represent individuals belonging to social communities [36].

Given a graph $G(\mathcal{N}, \mathcal{L})$, a community is a subgraph $G'(\mathcal{N}', \mathcal{L}')$, whose nodes are tightly connected, i.e., cohesive. A type of definition of community structures is based on the relative frequency of links. In this case communities are seen as groups of nodes within which connections are dense, and between which connections are sparser. An example is shown in Fig. 1. The simplest formal definition in this class has been proposed in [37]: G' is a community if the sum of all degrees within G' is larger than the sum of all degrees toward the rest of the graph.

In complex networks,inter-community communication depends on few key nodes when intra-community communication is frequent. The importance of each node in a community (such as *degree* and *betweenness*) is not equal. The key nodes among the community are not only more easily accessible by nodes within the community, but are also easier to access by the external nodes as well. If the weight of a link is taken into account, it is named as the weighted complex network. For example, as shown in Fig. 1, the entire topology can be regarded as one community if we do not consider the weight of the link. However, given that the network traffic behavior leads to the variation of the weight of the link, there are two obvious communities, where $v1$ and $v2$ are two key nodes for these two communities respectively.

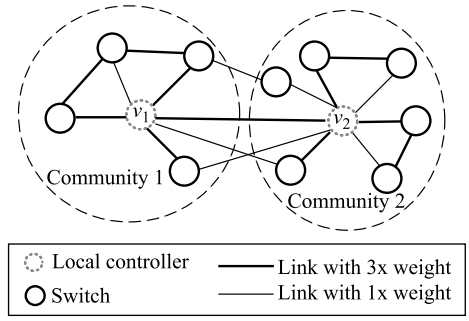

Fig. 1 Community in weighted complex network

When the community structure is unknown, modularity (abbreviated as Q) is the first and still currently the most popular measure of the quality of a partition into communities.

Q value suggested an alternative approach to finding community structure. Considering the computational complexity and performance, this paper used *Fast algorithm* [38] to achieve the community partition without overlap and loop, which is based on a standard "greedy" optimization algorithm.

The appendix of this paper introduce more detail about community theory, modularity Q and how to detect community structure in networks.

3.1.2 Selection of controllers

With the controller placement problem, the community theory is the motivation of our proposed approach. In SDN network, tasks can be divided into two types: local and global. Local tasks can be completed by a single local controller with the *local algorithm*. However, completing global task requires cooperation and communication between the local controllers. Therefore, some local controllers and some switches that communicated with each other construct one complex network. Wherein, the local controller and the

switch are the node of the network, and the communication between the nodes is the edge of the network. Given the unequal traffic of the edge, the weight of the edge is also not equal. Therefore, the network is a weighted complex network.

Therefore, we can divide the network into some communities, and each local controller governs switches within it. In other words, building a community in which the local controller is placed on the most important nodes (with the biggest *betweenness*). For example, in Fig. 1, $v1$ and $v2$ are two locations of the local controller within two communities. Thus, in the case the switch in the choice of mapping to the local controller can avoid the "detour", and can also achieve the load balance between local controllers. From another perspective, the scenario is equivalent to the global optimization for the global controller.

However, because traffic in DCN frequently fluctuates with both temporally and spatially, the weight of edge also frequently fluctuates. Considering the complexity of AAMcon, this paper simply assume that the weight of edge is equal.

3.1.3 NFV-based elastic controller pool

In this paper, the underlying SDN network infrastructure is abstracted to support the inter-connection between virtual network functions (VNFs) and with the end-points.

As shown as Fig. 2, in this study, the local controller is abstracted as one VNF which runs in a virtual machine (VM) virtualized from this switch using NFV technology. Some local controllers (i.e., VNF) logically formed with a controller pool, and they are actually collocated within their switch respectively. These local controllers and the global controller build a hierarchical distributed controller plane. To complete forwarding decisions, switch can request the local controller from this pool on demand, where dynamically adaptive matching between the demand and supply of VNF can achieve. VNFs can be inter connected with each other and with the end-points in a certain way (forwarding path) to achieve the desired overall end-to-end functionality. This is known as "service chain". Thus, VNFs can be distributed over several switches connected through multiple heterogeneous transport networks.

From the view of operation, *Nova* in OpenStack can be used to manage the VM in switch. An agent runs in each switch and controls the hypervisor used for creation/migration/deletion of the VMs. Finally, the connectivity between VMs within the switch is managed by *Neutron* in OpenStack. *Neutron* plugin can be used to provide full control of the switches through an SDN controller, and to disable reactive packet-in mechanism for unknown incoming connections.

3.2 System's design

Based on above mentioned ideas, the paper proposes AAMcon which consists of two parts: adaptively adjusting controllers (**AAcon**) and adaptively mapping for the controller (**AMcon**). AAMcon supports the following two main load adaptation operations: (1) For AAcon, if the aggregate load exceeds the maximum capacity of existing controllers, it increases the controller pool by adding new controllers, triggering switch migrations to utilize the new controller. Similarly, when the load falls below a particular level, it decreases the controller pool accordingly to consolidate switches onto

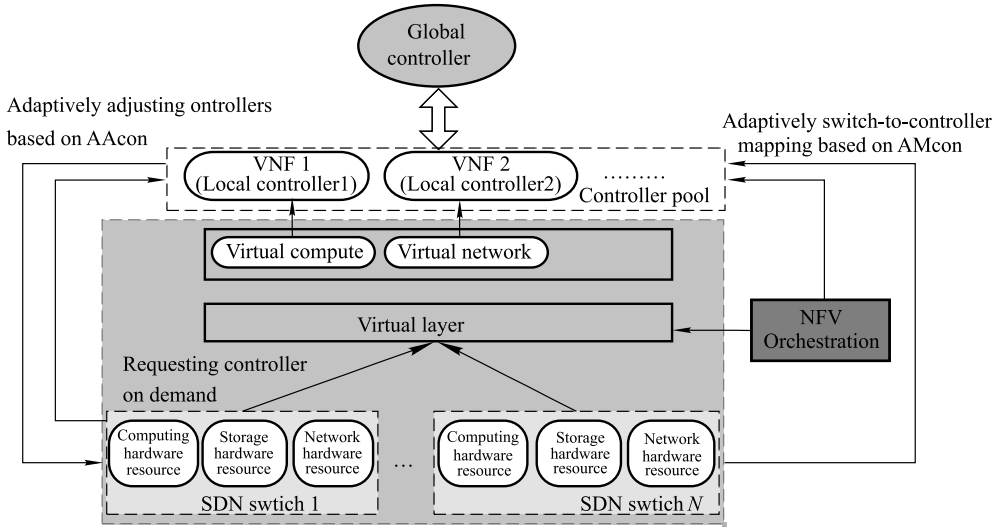

Fig. 2 NFV-based elastic controller pool

fewer controllers. (2) For AMcon, it monitors the load on all controllers and periodically balances the load of controllers by optimizing the switch-to-controller mapping.

3.2.1 AAcon—adaptively adjusting controllers

If the number of local controller can be adaptive to the traffic space-time distribution and is less occupied when the demand of switch is satisfied, then the controller resource can be used with the most efficiency. Thus, we propose the scheme of adaptively adjusting controllers—fast start/overload avoid algorithm. It is shown as Fig. 3. The scheme works mainly as follows: *Local_C* (local controller number) is initially set to a value, and gradually decreases according to demand. When *Local_C* decreases to a value in this case the load of the controller (*load_controller*) is overload, it resets to a new initial value. Then, one new descent process cycle starts, with more details presented in Algorithm 1.

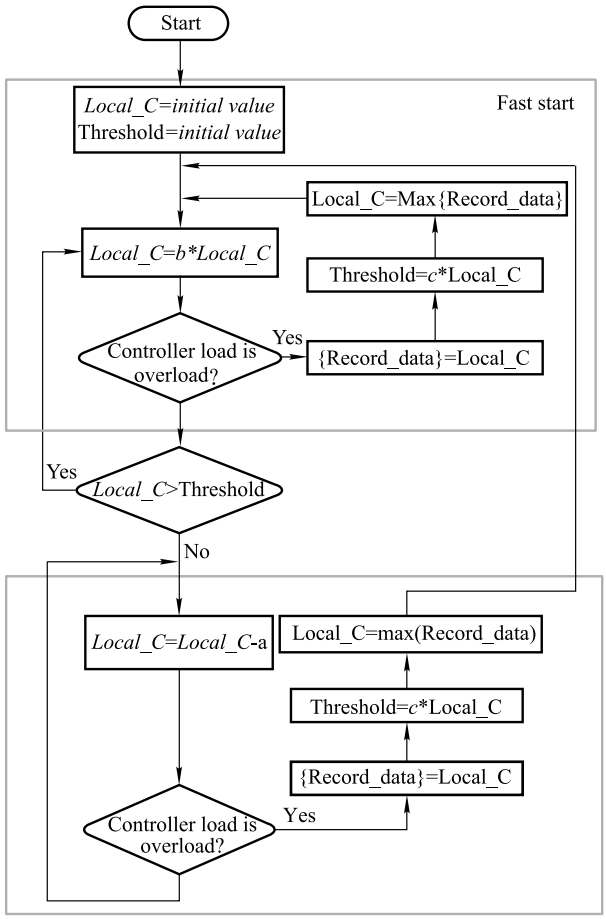

Fig. 3 Fast start/overload avoid algorithm

Lines 1–6: The controller does not overload, and then *Local_C*gradually decreases.

Lines 2–3: If the number of controllers does not decrease to the reserved threshold value (*Threshold*), after one round T, the *Local_C* exponentially reduces, where *round* () is the integral function.

Algorithm 1 Scheme of adaptively adjusting number of controller

```
1.  IF(load_controller<θ);
2.     IF(Local_C>Threshold);
3.     Local_C=round(b*Local_C);
4.     ELSE
5.     Local_C=Local_C-a;
6.     END IF
7.  ELSE
8.  Threshold=round(c*Local_C);
9.  Record_data(i, 1)=Local_C;
10. i = i + 1;
11. Local_C=max(Record_data);
12. END IF
```

Lines 4–5: If the number of controllers has reduced to a predetermined value, after one round T, the *Local_C* linearly reduces.

Lines 7–12: The controller overloads, and then the reserved threshold value (*Threshold*) is set to one new value (c*$Local\_C$). Then, the initial value of *Local_C* is set to the maximum value of past *Record_data*.

Line 8: The controller overloads, and then the reserved threshold value (*Threshold*) is set to c*$Local\_C$, where *Local_C* is the real-time value of the number of local controller when the controller overloads.

Lines 9–10: *Record_data* records the real-time value of the number of local controller when the controller overloads.

Line 11: Initial value of *Local_C* is set to the maximum value of past *Record_data*. This method can reduce the probability of excess controller load. It can automatically adapt the initial value of *Local_C* to the real network traffic values.

Where a, b, c are the regulation factors, and $0 < a$, $0 < b < 1$, $0 < c$. The initial value of T, θ, *Local_C*, *Threshold*, a, b, and c can be set as the following

T is one round time Algorithm 1, and its value does not affect the experiment result.

θ can be set according to the controller's carrying capacity and past experiment data. Without loss of generality, θ is set 90% of controller's carrying capacity in this paper.

The initial value *Local_C* can be set a relative larger value because the main idea of Algorithm 1 is that gradually decreasing according to demand. Its initial value does not affect the finally experiment result and only affect the convergence time because Algorithm 1 is adaptive-self.

Threshold, a, b and c is the intermediate variable in Algorithm 1. The initial valueof *Threshold* should be less than

the initial valueof *Local_C*, without loss of generality, and the initial valueof *Threshold* is set 65% of the initial valueof *Local_C* in this paper.

3.2.2 AMcon—adaptively mapping for controller

The concept in Section 3.1.2 is achieved based on heuristic, which is elaborated in Algorithm 2.

Algorithm 2 Scheme of mapping between switch and local controller

```
1. Community_result=Community_partion();
2. new_community=insert_controller();
3. While(load_controller>θ)
4. new_community=insert_controller();
5. Community_partion(new_community, 2);
6.    While(Q > 0.5 &&betweenness_controller is not the highest)
7.       Adding some edges and deleting other edges of community_1
         and community_2;
8.       betweenness_controller=compute_betweenness();
9.       Q=compute_Q();
10. End
11. End
```

Line 1: To an AS, the network is partitioned to some communities without overlap and loop by *Fast algorithm* [38].

Line 2: Adding one controller for each community, and this community is called new_community.

Lines 3–11: Continually adding controller to the new_community until the controller in the new_community is not overloaded, where θ is the threshold of determining whether controller is overload.

Line 4: Inserting one new controller to the new_community, and this community is called new_community.

Line 5: This new_community is partitioned into two communities, i.e., community_1 and community_2.

Lines 6–10: Q value of community_1 and community_2 should be more than 0.5, and the *betweenness* of the controller is not the highest in its community. If they are not satisfied, continually add and delete the edge of community_1 and community_2.

Line 7: To hold on the community characteristic ($Q > 0.5$), some edges are added between the inserted new controller and switches, correspondingly deleting the edge between the old controller and these switches. According to the greedy idea, adding or deleting some edges follow the direction of maximum increase or minimum decrease of the Q value of the community. The detailed discussion on the conditions that these actions (adding and deleting) should be executed and their method of execution can be found in [38].

Line 8: Computing the *betweenness* of the controller in community_1 and community_2.

Line 9: Computing the value of Q for community_1 and community_2;

3.2.3 Load measurement for controller

Algorithm 1 shows that *Threshold* can adjust the supply rhythm of the number of controllers, that is, it linearly decreases or exponentially decreases. The value of *Threshold*, which is determined when the controller overload, is a dynamic adaptive adjustment process. However, real-time demand for controllers determines whether the controller overload, therefore, the supply of the number of controllers is eventually regulated by its demand. This process then repeats itself, and the dynamic match between the demand and supply for the number of controllers can be finally completed. In this process, the real-time value of demands does not need to be measured. However, the load of controller (load_controller) must be measured.

The most direct method to measure the load of a controller is by sampling the response time of the controller at the switches. This response time include both computation and network latency. However, switches may not support response time measurements because this will require maintaining some amount of extra state at the switches that may or may not be feasible.

The experiments show that CPU utilization is roughly in proportion to the packet arrival rate [10]. Then, we can assume that the fraction of controller resources used by a switch is proportional to its fraction of the total packets received at the controller, which is typically true due to the almost linear relationship between throughput and packets. The packet arrival rate from each switch connected to the controller can help AAMcon in first dissecting the contribution of each switch to the overall CPU utilization.

Given that the controller is more programmable, AAMcon maintains a load measurement module on each controller to periodically report the CPU utilization and network I/O packet arrival rates at the controller.

Our experiments and reference [10] also show that the CPU is typically the throughput bottleneck for the controller. Thus, this paper uses CPU utilization to measure the load of a controller. The load measurement module averages load estimates over small time intervals (60s in this study) to avoid triggering switch migrations due to short-term load spikes.

3.3 Problem formulation and numerical analysis

Assuming that the topology of the network with 50 switch

nodes and the network's Q is 0.605. The distance between any two switches and the number of controllers required for the network (*Local_C*) are also known. The section analyzes the average distance between switches and controllers (abbreviated as *Lavg*) as shown in Eq. (1), and also find the relationship between the value of Q and average distance.

Considering that the physical network is modeled by a graph denoted by the tuple $G = (V, E, C)$, where $V = \{v_1 \cdots v_N\}$ is the set of switch nodes, E is the set of edges, and $C = \{c_1 \cdots c_w\}$ defines the set of controllers. The controller can be placed on any switch node in the network. $\{v_i, c_j\}$ represent the adjacency matrix between the switch node and the controller.

$$Lavg = \frac{1}{N}\sum_{i=1}^{N} d(v_i, c_j), \tag{1}$$

where $d(v_i, c_j)$ is the shortest distance (hops) between the switch and controller, and is computed by *Dijkstra* algorithm. We also assume that the distance between the controller and the switch where it is placed is 1.

Finding the controller placement c_j from the set of all possible controllers C with minimizing *Lavg* can be expressed as the following optimization problem:

Min $Lavg = \frac{1}{N}\sum_{i=1}^{N} d(v_i, c_j)$.

$S.t.$

(a) $Local\_C = F$, the number of controllers is constant.

(b) $\sum_{i=1}^{N}\{v_i, c_j\} > 0, \forall j \in w$, each controller connects to at least one switch node.

(c) $\sum_{j=1}^{k} d\{v_i, c_j\} = 1, \forall i \in N$, each switch can only connect one controller.

(d) $Max\{d(v_i, c_j)\} \leqslant D_{req}$, guaranteeing QoS for distance.

(e) $Max\{\text{load\_controller}(c_j)\} < \theta, \forall j \in w$, guaranteeing each controller not to be overloaded.

D_{req} is the required maximum threshold of distance for QoS, and load_controller(c_j) is the load of the controller c_j.

3.3.1 Comparison of AMcon's and optimal value for Lavg

We use the Genetic Algorithm to solve the above optimization problem in the following two cases: 1) The optimal solution of *Lavg* ($Lavg_{\min}$, i.e., optimal value of *Lavg*). 2) The optimal solution of *Lavg* for AMcon, where the controller is placed on the switch with the largest *betweenness* among its community to add the constraints of $Q > 0.5$.

The analysis results are shown in Table 2, where $Lavg_{\min}$ is the optimal value of *Lavg* without $Q > 0.5$. For AMcon, we let $Q > 0.5$ to guarantee the community characteristics. From Table 2, we found that the gap between AMcon and the optimal value is small.

Table 2 Least *Lavg* of AMcon and optimal value

	Number of controllers (*Local_C*)	AMcon	$Lavg_{\min}$
Lavg	2	2.52	2.32
	4	2.44	2.28

3.3.2 Relationship between the Q value and average distance

Under the abovementioned conditions with *Local_C*=3, we allowed the 50 switches to arbitrarily connect to any of the 3 controllers to find the relationship between the value of Q and average distance according to the full combination results between the switch and controller. From the result, we can plot the blue line of Fig. 4, where the horizontal axis is the average distance with hops.

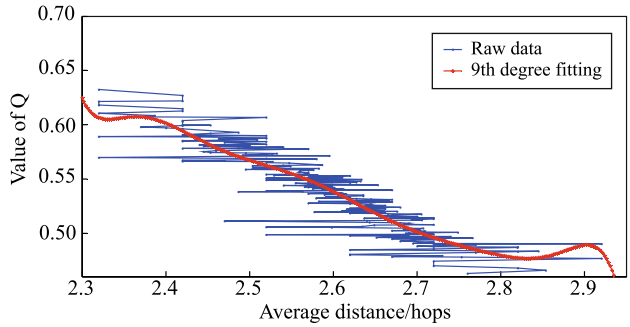

Fig. 4 Relationship between the Q value and average distance

The following conclusions can be drawn from the blue line of Fig. 4: First, the correspondence between the Q value and average distance is not one-to-one, but a corresponding range exists, where in a certain network community structure (i.e., a Q value), its average distance fluctuates in a certain range. Second, the trend shows that the Q value and average distance is indeed an inverse relationship, i.e., the average delay decreases when the Q value increases. This observation in in good agreement with the experimental results (as described later). Furthermore, we carried out the trend of 9th degree fitting, with results following the norm of residuals = 0.24547 to Eq. (2),

$$\begin{aligned} y = &-3e + 4x^9 + 7e + 4x^8 - 7.3e + 6x^7 + 4.4e + 7x^6 \\ &-1.7e + 8x^5 + 4.5e + 8x^4 - 7.9e + 8x^3 \\ &+8.8e + 8x^2 - 5.7e + 8x + 1.7e + 8, \end{aligned} \tag{2}$$

where y is the Q value and x is the arerage distance with hops. We plot the red line of Fig. 4.

The above analysis has shown that AMcon is a practical solution. Therefore, in the actual deployment process, we can try to let the SDN network hold an obvious community characteristic where some communities are built, and then the

controller is placed on the switch with the largest *betweenness* among its community. Then, its performance can be approximated to the optimal value $Lavg_{\min}$.

4 Experiments

We evaluate the performance of our proposed framework through extensive simulations. We have deployed networks with different sizes and resource requirements to test our current implementation on a real test-bed: distributed OpenFlow testbed (DOT) [39].

Bari et al. [40] presented an algorithm with dynamically and efficiently provision controllers in a WAN by periodically reassigning switches to controllers. It combines Greedy Knapsack with Simulated Annealing heuristic. We just use the greedy algorithm of *DCP-GK* since the full algorithm is designed for controller provisioning in WANs where time complexity is less of an issue. *DCP-GK* is a typical adaptive solution.

Jiménez et al. [27] proposed the *k-critical* algorithm, which identifies the minimum number of controllers and their location to create a robust control topology that deals robustly with failures and balances the load among the selected controllers. On the other hand, *k-critical* is a typical static solution.

Thus, the proposed scheme (abbreviated as *AAMcon*) is compared with *DCP-GK* and *k-critical*.

4.1 Experiment setup

1) Topologies: we conduct our simulations using the widely adopted fat-tree topologies. For fat-tree, the number of pods is 24 with a total of 3,456 hosts and 720 switches.

2) Flow generation model: the flow generation pattern follows the *Zipf* parameters, which are of the form, where $Pr\{C_k\}$ is the probability of the kth most routers being selected as the destinations and $\alpha = 0.7$. We use *iperf* to generate TCP flows between the end hosts. The end hosts of each flow is chosen randomly. To make the traces more realistic, we generate the flows according to the distribution of flow size, flow inter arrival times, and number of concurrent flows reported in a recent study on network traffic characterization [41]. The generated traffic spans 48 hours capturing the time-of-day effect.

4.2 Experiment results

This section presents the experiment results. Unless otherwise stated, each experimental point reported is averaged over 10 experimental runs.

4.2.1 Load adaptation

The primary objective of the proposed scheme is to determine whether the number of controllers can dynamically increase or decrease according to the significant variations in both spatial and temporal traffic, which results in the variations of the demanded number of controllers. Figure 5 shows the variations of the provided and demanded controllers with time.

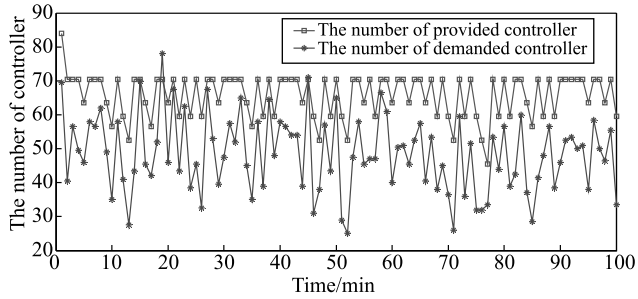

Fig. 5 Relationship between the provided and demanded controllers for AAMcon

In Fig. 5, the provided controllers of *AAMcon* can greatly follow the demanded controllers, and their relation coefficient is close to 0.8. During the whole experiment, the number of demanded controllers over the provided controllers is only 2 times, i.e., the *success rate* is 98%. We let *overflow rate* to reflect the cost of the provided controller satisfying the demand, which is defined as . Where $n_1(t)$ is the number of provided controller, and $n_2(t)$ is the number of demanded controller at time t. Experiment results show that the average value of the *overflow rate* with 27.8% is not high to achieve a *success rate* at 98%. The *success rate* and *overflow rate* is dependent, and higher *overflow rate* can result in higher *success rate*.

4.2.2 Number of controllers for average delay

Unless otherwise stated, the *delay* in this study represents the switch-to-controller propagation delay whose unit is μs. In the experiments, we gradually increase the number of controllers for each placement strategy until no overload occurred.

As shown in Fig. 6, the horizontal axis is the average delay, and the vertical axis is the number of controllers. We observe some trends in the results. First, as expected, more controllers result in less average delay. Second, the reduction of average delay does not lineally follow the increase of the number of controllers. For example, the delay is almost unchanged when the number of controllers increases from 8 to 16 for all schemes. The efficiency with this increase is

very low. Particularly, there should be an optimal number of controllers that adding another controller results in negligible delay reduction. BC= $\frac{\Delta t}{\Delta N}$ (Benefit/Cost) is defined to find it, where is the reduction of delay when $\Delta t(us)$ is the increase of the number of controllers. Figure 6 shows that the optimal number of controllers is 8 when we consider BC > 0.1. A significant delay range is covered when 8 controllers are considered. The delay ranges are 4.24μs–3.34μs, 4.46μs–3.5μs, and 6.75μs–5.0μs for *AAMcon*, *DCP-GK*, and *K-critical*, respectively. We can see that two adaptive solutions outperform the static approach with a large advantage.

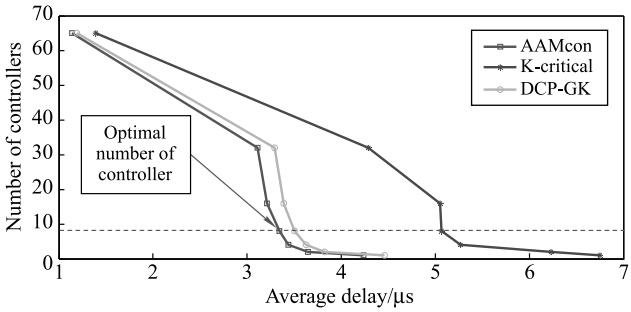

Fig. 6 Number of controllers for average delay

In summary, although numerical analysis and experiment results have confirmed that more controllers can decrease the delay, we need the tradeoff between the number of controllers and delay.

4.2.3 CDF of the minimal required controllers to avoid overload

During the experiment, placement quality varies widely. A few placements are pathologically bad, while most are mediocre, and only a small percentage is optimal. From another perspective, Fig. 7 shows the distribution of delay, which covers the number of controllers k = [65, 32, 16, 8, 4, 2, 1] for *AAMcon*, *DCP-GK,* and *K-critical,* respectively, where the vertical axis is the CDF (cumulative distribution function) of the delay. Regardless of the number of controllers, the delay of *AAMcon*is less than that of *DCP-GK* and *K-critical* when the number of controllers is the same. With average latency, *K-critical* is 1.2×–1.7× larger than that of *AAMcon*. When *DCP-GK* is used,it is 1.04×–1.06× larger than that of *AAMcon*, as shown in Fig. 7. It also confirms the other conclusion that *AAMcon* and *DCP-GK* are almost cross with close performance when k = 32 and k = 16.

From the above analysis, we can conclude that *AAMcon* utilizes a lower number of controller and ensures their higher utilization than the *DCP-GK* and *K-critical*, which reduces the overall system cost and energy consumption extensively.

4.2.4 Failure tolerance

Although the primary objective of our scheme is the load adaptation, another aspect should be concentrated on failure tolerance to confirm its scalability.

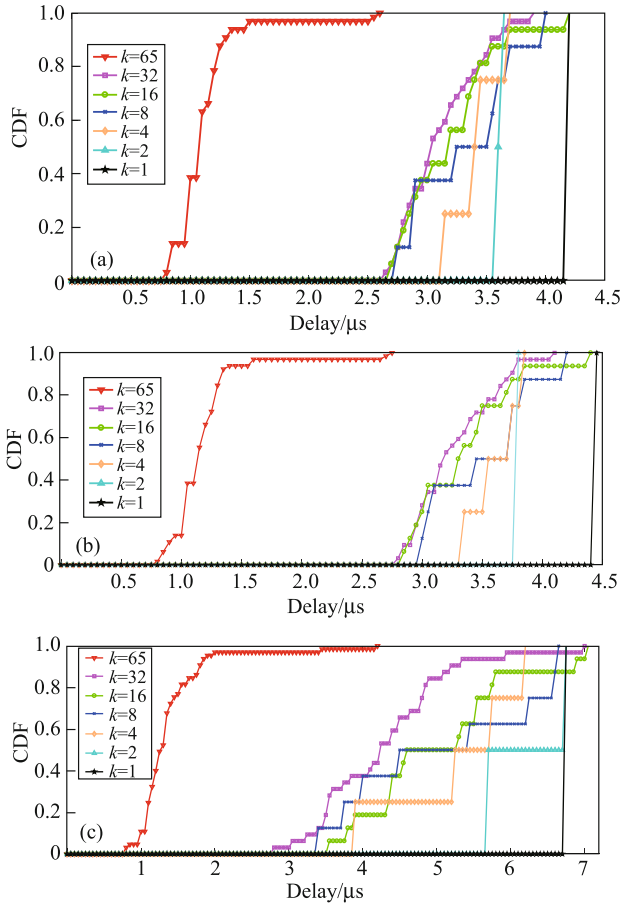

Fig. 7 Delay CDFs for all possible controller combinations for k = [65, 32, 16, 8, 4, 2, 1]

To evaluate the performance of the proposed scheme for network fault, we randomly let the 3n link among the experiment network topology to be invalid, where n is the number of controllers. The number of fault link increases with the increase in the number of controllers. The experiment aims to observe the self-adaptive fault recovery of proposed scheme when the same number of links are failed.

Figure 8 plots the obtained experiment results, where the vertical axis is the increased average delay due to failure. As shown in Fig. 8, as expected, the delay increase due to fault correspondingly increased. However, with lower increased average delay for percent, *AAMcon* shows the advantage compared to *DCP-GK* and *k-critical* for different cases. Particularly, *AAMcon* shows better robustness. This benefit

from efficient switch-to-controller mapping that *AAMcon* can adaptively adjust the mapping for switch-to-controller as presented as Section 3.2.2. Therefore, when the link of a switch-to-controller, *AAMcon* can let the switch to self-adaptive select other controller through other links.

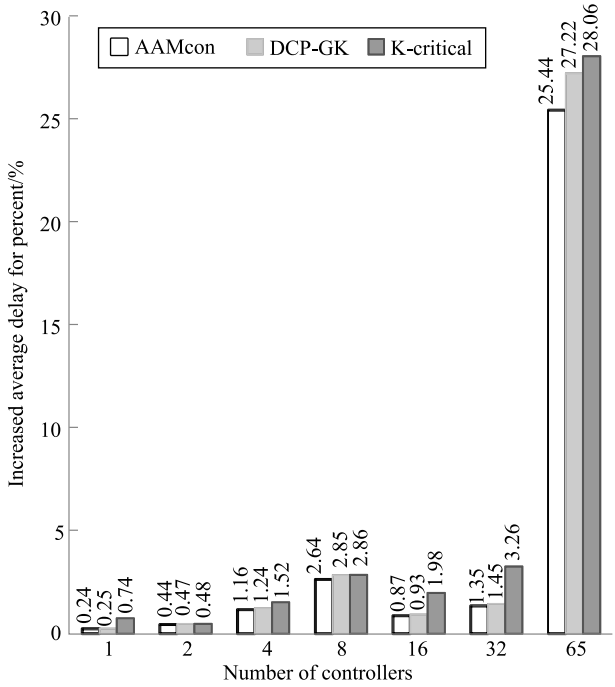

Fig. 8 Increased average delay due to failure

4.2.5 The load balance of controller

In this section, we analyze the load balance of each controller in the network. To measure the load balance of each controller, we introduce the Gini coefficient of the controller load (Load_G), which is defined as:

$$Load\_G = \frac{\sum_{i=1}^{N}\sum_{j=1}^{N}|y_i - y_j|}{2N^2\overline{y}}, \tag{3}$$

where y_i is the load of the controller i, $\overline{y}$ is the average of the controller load, and N is the number of controllers. Higher Load_G denotes highly unbalanced load.

Figure 9 shows Load_G in a variety of experimental scenario for *AAMcon*, *DCP-GK* and *k-critical*, where the horizontal axis is the number of controllers. The Gini coefficient of AAMcon decreases slightly with the increase of the number of controllers. Its low Gini coefficient indicates that AAMconcan achieve the load balance between the controllers. On the other hand, *DCP-GK* is also outperform *k-critical*. It means that two adaptive solutions can provide more load balance than the static approach with a large advantage. Particularly, *AAMcon* can provide the most performance among them. This thanks to that *AAMcon* can adaptively adjust the mapping for switch-to-controller and the number of controller as presented as Section 3.2. Therefore, the load can be evenly assigned to controllers.

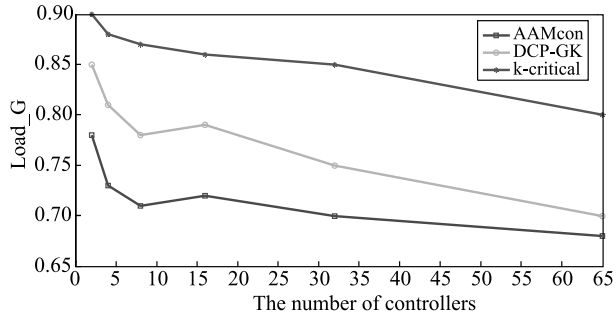

Fig. 9 Gini coefficient of controllers

4.2.6 Community modularity affects the delay

Well known, higher Q value denotes higher significance of the community structure as presented as 3.1. To a network with same number of edges and vertices, the average delay decreases when the Q value increases. This paper also uses Q value to observe how network topology with different community structure affects the delay between the switch and controller.

From Fig. 10, as expected, the average delay decreases when the value of Q increases, which indicates that the *AAMcon* scheme can show better performance to the network with more significant community structure. On the other hand, it also gives us one method to improve the performance of SDN controller by adding or deleting edges to improve the Q value when an SDN network is built. Besides, the delay is sensitive to the change of Q value, and is reduced from 4.26 to 3.87 (with 10%) when the Q value increased from 0.904 to 0.92 (with 1.75%).

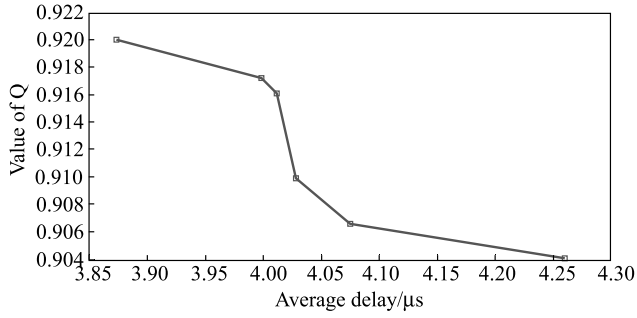

Fig. 10 Average delay with the value of Q

5 Discussion

1) Controller migration/scaling

We have addressed some key challenges of AAMcon. The switch migration between controllers is closely related to this study. This is especially sensitive for in-network solutions since the protocol packets shares the same links with traffic load.

Migrating a switch from one controller to another in a naive fashion can cause disruption in ongoing flows, which can severely affect the various running applications. Thus, certain configurations should be made in other switch nodes before moving a controller elsewhere. In this case OpenFlow *packet-in* is send to the right controller. This can maintain the flow continuity while migrating a controller.

Dixit et al. [10] have introduced a novel *4-phase* switch migration protocol to enable the load shifting. The state consistency during the switch handover is guaranteed by a protocol similar to a 2-phase commitment. Such a modified protocol can be used in our study and the protocol for inter-controller communication is outside the scope for this paper.

2) The average distance between switches and controllers

We have conducted the analysis for the average distance between switches and controllers (*Lavg*) where the number of controllers required for the network (*Local_C*) is assumed to be known. If *Local_C* is not known, then this becomes a multi-objective optimization problem as follows:

Min $\{Lavg = \frac{1}{N}\sum_{i=1}^{N} d(v_i, c_j), Local\_C\}$,

s.t.

(i) $\sum_{i=1}^{N}\{v_i, c_j\} > 0, \forall j \in w$, each controller connects to at least one switch node.

(ii) $\sum_{i=1}^{N}\{v_i, c_j\} = 1, \forall i \in N$, each switch can only connect one controller.

(iii) $Max\{d(v_i, c_j)\} \leqslant D_{req}$, *Qos* guarantee.

(iv) $Max\{\text{load\_controller}(c_j)\} < \theta$, $\forall j \in w$, guaranteeing each controller is not overloaded.

For multi-objective problems, we cannot identify a single solution that simultaneously optimizes each objective. A tentative solution is called Pareto optimal set if it cannot be eliminated from consideration by replacing it with another solution which improves an objective without worsening another one.In practice, we can select one appropriate solution from this set according to some principles.

3) The topics explored in the future

First, the community characteristic in a real network is common, whether there is a relationship between the number of controllers and the Q value. If this relation exists, then we can reduce the number of controllers by adjusting the value of Q.

Second, the information-centric networking (ICN) is another future network architecture, where under the combination of SDN and ICN, multi-source transmission at the network-layer can improve transmission efficiency [42].

Third, we have conducted the analysis for AMcon. On the other hand, AAcon can be considered as a variant of TCP *Slow-Start* algorithm for SDN. Therefore, we can also conduct the theoretical analysis for AAcon [43].

6 Conclusion

With distributed SDN controllers in data center networks, this study proposed AAMcon to solve the following problems: very slow reaction time between the switch and controller, the difficulty of the control plane in adapting to traffic load variations for real networks, and the difficulty of selecting controllers. The experiments have confirmed the following conclusions: 1) with the number of controllers, the provided controllers can greatly follow the demand, and more controllers can reduce the delay between the switch and controller. 2) On the other hand, using more than the optimal number of controllers can be inefficient and costly because the delay improvement is negligible. 3) With failure tolerance, AAMcon shows good robustness. 4) With the delay, AAMcon scheme can show better performance to the network with more significant community structure. In fact, there is an inverse relationship between the Q value and average distance, i.e., the average delay decreases when the Q value increases.

Acknowledgements This work was supported by the National Natural Science Foundation of China (Grant Nos. 61872102, 61571141, 61802080, 51575116, 61806058), the China National Spark Program (2015GA780065), the Science and Technology Project of Guangdong Province (2017A010102014, 2016A010102022); the Innovative Team Project of Guangdong Universities (2017KCXTD025); the Innovative Academic Team Project of Guangzhou Education System (1201610013), and Guangzhou Science and Technology Project (201604016001), China.

Appendix

1) Community theory and modularity Q

Within a network with m edges and n vertices, we consider a partition of a network into u communities. Let us define a $u \times u$ symmetric matrix E whose element e_{ij} is the fraction of all edges in the network that link vertices in community i to vertices in community j. Modularity Q is defined in the

following way [37],

$$Q = \sum_i (e_{ii} - a_i^2) = \mathrm{Tr}E - \|E^2\|. \quad (4)$$

The trace of this matrix $\mathrm{Tr}E = \sum_i e_{ii}$ is the fraction of edges in the network that connect vertices in the same community, while the row (or column) sums $a_i = \sum_j e_{ii}$ give the fraction of edges that connect to vertices in community i. $\|E^2\|$ indicates the sum of the elements of the matrix E^2. The Q measures the degree of correlation between the probability of having an edge joining two sites and the fact that the sites belong to the same community. Higher Q value denotes higher significance of the community structure. $Q = 1$, which is the maximum, indicate a strong community structure; conversely, $Q = 0$ means a random graph with no community structure. In practice, Q value is greater than about 0.3 appear to indicate significant community structure. In weighted complex network as shown as Fig. 1,the link weight also affects the Q value.

This paper adopted *betweenness* to measure the importance of node in a community. As with the reference [37], the node *betweenness* is defined as follows:

$$Betw(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}, \quad (5)$$

where σ_{st} is the total number of shortest paths from node s to node t, and $\sigma_{st}(v)$ is the number of those paths that pass through node v.

2) Detecting community structure in networks

On a network with m edges and n vertices, *Fast algorithm* is the following:

Starting with a state in which each vertex is the sole member of one of n communities, we repeatedly join communities together in pairs, choosing at each step the join that results in the greatest increase (or smallest decrease) in Q. The progress of the algorithm can be represented as a "dendrogram," a tree that shows the order of the joins. Cuts through this dendrogram at different levels give divisions of the network into larger or smaller numbers of communities and we can select the best cut by looking for the maximal value of Q.

Since the joining of a pair of communities between which there are no edges at all can never result in an increase of Q, only those pairs between which there are edges need to be considered, of which there will at any time be at most m, where m is the number of edges in the graph. The change in Q upon joining two communities is given by $\Delta Q = e_{ij} + e_{ji} - 2a_i a_j = 2(e_{ij} - a_i a_j)$, which can clearly be calculated in constant time. Following a join, some of the matrix elements e_{ij} must be updated by adding together the rows and columns corresponding to the joined communities, which takes worst case time $O(n)$. Thus each step of the algorithm takes worst-case time $O(m + n)$. There are a maximum of $n - 1$ join operations necessary to construct the complete dendrogram and hence the entire algorithm runs in time $O((m + n)n)$, or $O(n^2)$ on a sparse graph. The algorithm has added advantage of calculating the Q value as it goes along, making it especially simple to find the optimal community structure.

References

1. McKeown N, Anderson T, Balakrishnan H. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69–74
2. Kreutz D, Ramos F M V, Verissimo P E, Rothenberg C E, Azodolmolky S, Uhlig S. Software-defined networking: a comprehensive survey. Proceedings of the IEEE, 2015, 103(1): 14–76
3. Jain S, Kumar A, Ong J, Poutievski L, Singh A, Zolla J. B4: experience with a globally-deployed software defined WAN. ACM SIGCOMM Computer Communication Review, 2013, 43(4): 3–14
4. Al-Fares M, Radhakrishnan S, Raghavan B, Huang N, Vahdat A. Hedera: dynamic flow scheduling for data center networks. In: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, 2010, 89–92
5. Canini M, Kuznetsov P, Levin D. A distributed and robust SDN control plane for transactional network updates. In: Proceedings of the 34th Annual IEEE International Conference on Computer Communications. 2015, 190–198
6. Koponen T, Amidon K, Balland P, Casado M, Chanda A, Fulton B, Lambeth A. Network virtualization in multi-tenant datacenters. In: Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation. 2014, 203–216
7. Cao X, Popescu I, Chen G, Guo H, Yoshikane N, Tsuritani T, Morita I. Optimal and dynamic virtual datacenter provisioning over metro-embedded datacenters with holistic SDN orchestration. Optical Switching and Networking, 2017, 24: 1–11
8. Wang T, Liu F, Guo J, Xu H. Dynamic SDN controller assignment in data center networks: stable matching with transfers. In: Proceedings of the 35th Annual IEEE International Conference on Computer Communications. 2016, 1–9
9. Benson T, Akella A, Maltz D. Network traffic characteristics of data centers in the wild. In: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement. 2010, 267–280
10. Dixit A, Hao F, Mukherjee S, Lakshman T V, Kompella R R. ElastiCon: an elastic distributed SDN controller. In: Proceedings of 2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems. 2014, 17–27
11. Liu J, Panda A, Singla A, Godfrey B, Schapira M, Shenker S. Ensuring connectivity via data plane mechanisms. In: Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation. 2013, 113–126
12. Newport C, Zhou W. The (surprising) computational power of the SDN

data plane. In: Proceedings of the 34th Annual IEEE International Conference on Computer Communications. 2015, 496–504

13. Operators N. Network functions virtualization: an introduction, benefits, enablers, challenges & call for action. In: Proceedings of SDN and OpenFlow SDN and OPenFlow World Congress. 2012
14. Muñoz R, Vilalta R, Casellas R. Integrated SDN/NFV management and orchestration architecture for dynamic deployment of virtual SDN control instances for virtual tenant networks. Journal of Optical Communications and Networking, 2015, 7(11): B62–B70
15. Levin D, Wundsam A, Heller B, Handigol N, Feldmann A. Logically centralized? State distribution trade-offs in software defined networks. In: Proceedings of the 1st workshop on Hot Topics in Software Defined Networking. 2012, 1–6
16. Nguyen X N, Saucez D, Barakat C. OFFICER: a general optimization framework for OpenFlow rule allocation and endpoint policy enforcement. In: Proceedings of the 34th Annual IEEE International Conference on Computer Communications. 2015, 478–486
17. Bianchi G, Bonola M, Capone A. OpenState: programming platform-independent stateful openflow applications inside the switch. ACM SIGCOMM Computer Communication Review, 2014, 44(2): 44–51
18. Moshref M, Bhargava A, Gupta A. Flow-level state transition as a new switch primitive for SDN. In: Proceedings of the ACM Special Interest Group on Data Communication. 2014, 61–66
19. Schmid S, Suomela J. Exploiting locality in distributed SDN control. In: Proceedings of the 2nd ACM Special Interest Group on Data Communication Workshop on Hot Topics in Software Defined Networking. 2013, 121–126
20. Vissicchio S, Tilmans O, Vanbever L. Central control over distributed routing. In: Proceedings of the 2nd ACM Special Interest Group on Data Communication Workshop on Hot Topics in Software Defined Networking. 2015, 43–56
21. Krishnamurthy A, Chandrabose S P, Gember-Jacobson A. Pratyaastha: an efficient elastic distributed SDN control plane. In: Proceedings of the 2nd ACM Special Interest Group on Data Communication Workshop on Hot Topics in Software Defined Networking. 2014, 133–138
22. Xie J, Guo D, Hu Z. Control plane of software defined networks: a survey. Computer Communications, 2015, 67: 1–10
23. Yao G, Bi J, Li Y, Guo L. On the capacitated controller placement problem in software defined networks. IEEE Communications Letters, 2014, 18(8): 1339–1442
24. Sallahi A, St-Hilaire M. Optimal model for the controller placement problem in software defined networks. IEEE Communications Letters, 2015, 19(1): 30–33
25. Ros F J, Kuiz P M. On reliable controller placements in software defined networks. Computer Communications, 2015, 77: 41–51
26. Jiménez Y, Cervelló-Pastor C, Garcia A. Dynamic resource discovery protocol for software defined networks. IEEE Communications Letters, 2015, 19(5): 743–746
27. Jiménez Y, Cervelló-Pastor C, GarciaA J. On the controller placement for designing a distributed SDN control layer. In: Proceedings of IFIP Networking Conference. 2014, 1–9
28. Lange S, Gebert S, Zinner T. Heuristic approaches to the controller placement problem in large scale SDN networks. IEEE Transactions on Network and Service Management, 2015, 12(1): 4–17
29. Matias J, Garay J, Toledo N. Toward an SDN-enabled NFV architecture. IEEE Communications Magazine, 2015, 53(4): 187–193
30. Gember-Jacobson A, Viswanathan R, Prakash C. OpenNF: enabling innovation in network function control. ACM SIGCOMM Computer Communication Review, 2014, 44(4): 163–174
31. Batalle J, Riera J F, Escalona E. On the implementation of NFV over an OpenFlow infrastructure: routing function virtualization. In: Proceedings of 2013 IEEE SDN for Future Networks and Services. 2013, 1–6
32. Wang L, Anta A F, Zhang F, Wu J, Liu Z. Multi-resource energy-efficient routing in cloud data centers with networks-as-a-service. In: Proceedings of 2015 IEEE Symposium on Computers and Communication. 2015, 694–699
33. Brief O N F S. OpenFlow-enabled SDN and network functions virtualization. Open Netw. Found, 2014
34. Rodriguez-Natal A, Ermagan V, Noy A. Global state, local decisions: decentralized NFV for ISPs via enhanced SDN. IEEE Communications Magazine, 2017, 55(4): 87–93
35. Filiposka S, Juiz C. Community-based complex cloud data center. Physica A: Statistical Mechanics and its Applications, 2015, 419: 356–372
36. Boccaletti S, Latora V, Moreno Y. Complex networks: structure and dynamics. Physics Reports, 2006, 424(4): 175–308
37. Radicchi F, Castellano C, Cecconi F. Defining and identifying communities in networks. Proceedings of the National Academy of Sciences, 2004, 101(9): 2658–2663
38. Newman M E J. Fast algorithm for detecting community structure in networks. Physical Review E, 2004, 69(6): 066133
39. Roy A R, Bari M F. Design and management of DOT: a distributed OpenFlow testbed. In: Proceedings of IEEE/IFIP Network Operations and Management Symposium. 2014, 1–9
40. Bari M F, Roy A R, Chowdhury S R. Dynamic controller provisioning in software defined networks. In: Proceedings of IEEE International Conference on Network and Service Management. 2013, 18–25
41. Gebert S, Pries R, Schlosser D. Internet access traffic measurement and analysis. In: Proceedings of International Workshop on Traffic Monitoring and Analysis. 2012, 29–42
42. Liu W X, Yu S Z, Tan G. Information-centric networking with built-in network coding to achieve multisource transmission at network-layer. Computer Networks, Elsevier, 2017, 115(3): 110–128
43. Liu W X, Zhang J, Liang Z W, Peng L X. Content popularity prediction and caching for ICN: a deep learning approach with SDN. IEEE Access, 2018, 6: 5075–5089

Waixi Liu received the PhD in communication and information system from Sun Yat-Sen University, China in 2013. He is currently an associate professor in the Department of Electronic and Information Engineering, Guangzhou University. His research interests are in future network, data mining and network coding. He has published more than 20 papers.

Yu Wang received the PhD degree in computer science from Deakin University, Victoria. He is currently associate professor with Guangzhou University, China. His research interests include network traffic modeling and classification, social networks, mobile networks, and network security.

Jie Zhang received the MS degree in electrical engineering from South China University of Technology, China in 2006. Presently, he is an engineer in the School of Mechanical and Electrical Engineering at Guangzhou University, China. His research field is distribution system reliability evaluation.

Hongjian Liao is a PhD candidate of South China Normal University, China and he is currently associate research fellow with Guangzhou University, China. His research interests include knowledge recommendation based on context-aware, e-learning and blended learning.

Zhongwei Liang received the PhD degrees from South China University of Technology, China in 2010. He now is a professor in Guangzhou University, China. His current research interests include deep learning, intelligent manufacturing and image processing.

Xiaochu Liu received the PhD degrees from South China University of Technology, China in 2006. He now is a professor in Guangzhou University, China. His current research interests include robotics, intelligent manufacturing, and the resource allocation in next-generation wireless communication systems.