Computer Communications 000 (2016) 1–10

Contents lists available at ScienceDirect

Computer Communications

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

Adaptive Distributed Software Defined Networking

Yanyu Chen[a,b], Yuan Yang[b], Xiaoyue Zou[a], Qi Li[a,c,*], Yong Jiang[a]

[a] Graduate School at Shenzhen, Tsinghua University, Shenzhen 518055, PR China
[b] Department of Computer Science, Tsinghua University, Beijing 100084, PR China
[c] Tsinghua National Laboratory for Information Science and Technology(TNList), Tsinghua University, Beijing 100084, PR China

ARTICLE INFO

Article history:
Received 28 April 2016
Revised 16 September 2016
Accepted 21 November 2016
Available online xxx

Keywords:
SDN
Algorithm
Controller selection problem

ABSTRACT

Distributed Software Defined Networking (SDN) federates multiple controllers in a network to solve the problems in single controller networks, e.g., to improve network reliability and reduce the delay between controllers and switches. However, in the current distributed SDN schemes, the mapping between SDN switches and controllers is statically configured, which may result in uneven load distribution among controllers. These schemes cannot fully benefit from the distributed SDN architecture. In order to address this issue, this paper proposes ESDN, an *adaptive elastic distributed SDN architecture*. The architecture dynamically selects a minimum number of active controllers that switches attached to, and changes the mapping between switches and controllers according to the network load. Specially, a switch can migrate from one controller domain to another so that the mapping is adaptive to the network load. We formalize the controller selection problem as an optimization problem, and prove that the problem is NP-Hard. We solve the problem by using offline and online algorithms, respectively. With the heuristics, controllers in a network are dynamically changed with respect to the network load. The offline algorithm has an approximation ratio of 2 related to the optimal result, and the online algorithms can find similar number of active controllers within a shorter time. We validate the algorithms and evaluate the performance by simulations. In particular, the number of inactive controllers computed by shrinking action of online algorithm averagely achieves around 92% of the optimal values when the whole network load decreases from 65% controller capacity to 25% controller capacity.

1. Introduction

Software Defined Networking (SDN) [1,2] enables network programmability and easy management [3–5]. Since it achieves a centralized control plane architecture, it brings up some issues of scalability and reliability. Multiple distributed SDN controllers [6–8] are proposed to address these issues. Most of existing work focuses on the state consistency issue among multiple controllers. The mapping between switches and controllers is statically configured, which may result in uneven load distribution among controllers and controller crash by packet burst. For instance, the peak-to-median radio of traffic can be almost 1–2 orders of magnitude (see more detailed data in [9]). Therefore, these approaches cannot fully benefit from the distributed architecture.

In order to address the issues above, Dixit et al. [10] proposed an elastic distributed SDN controller scheme that dynamically shrinks and expands the controller pool with respect to the network load. To realize the elastic distributed SDN controller, a switch migration protocol is proposed to enable a switch migration from one controller domain to another. Unfortunately, the design only includes a basic elastic SDN controller architecture and a migration proposal. It does not discuss the key question in the elastic architecture, i.e., how to make a controller be adaptive to runtime network load with changes of network load and topology. The key challenge in the distributed SDN architectures remains unresolved. If switches cannot correctly select controllers for migration with network changes, namely *controller selection problem*, the scheme will still fail in balancing the load between controllers.

In this paper, we formalize the controller selection problem as an optimization problem in which switches can migrate to different controllers with respect to the network load, and the controllers can be activated and inactivated. We prove that the controller section problem is NP-Hard. We first solve the problem by using an offline algorithm. The offline algorithm selects controllers for each switch, which is a sub-optimal algorithm with an approximation ratio of 2. Then, we propose an adaptive Elastic distributed SDN architecture (ESDN), in which controllers are selected dynamically according to network load. When the whole network load falls below a given lower threshold, switches migrate to re-selected controllers to reduce the number of active controllers. After switch migrations, controllers without any load can be inacti-

* Corresponding author.
E-mail address: liqi@csnet1.cs.tsinghua.edu.cn (Q. Li).

http://dx.doi.org/10.1016/j.comcom.2016.11.009
0140-3664/© 2016 Published by Elsevier B.V.

vated. Once the loads of controllers exceed a given upper threshold, some switches attached to them migrate to re-selected controllers that have less load or that are inactive. Thereby, the controller pool is dynamically shrinking and expanding. We develop online algorithms to re-select controllers, such that only a part of switches need to migrate. Note that, since switches change their corresponding controllers only when there is a controller crash or significant control load change, the network stability is ensured with the proposed distributed architecture.

In summary, this paper makes the following contributions:

- We formally define the problem of controller selection problem and prove that the controller selection problem is NP-Hard. We develop an offline algorithm to solve the problem sub-optimally.
- We propose ESDN, an adaptive elastic distributed SDN architecture to select active controllers dynamically according to network load. We develop online algorithms to solve the controller selection problem with little overhead.
- We use simulations to evaluate the performance of the offline and online algorithms. The experiment results show that our algorithms can effectively shrink and expand the controller pool in the networks. The number of inactive controllers computed by shrinking action by online algorithm averagely achieves around 92% of the optimal values. And the expanding action by online algorithm can dynamically add necessary number of controllers to prevent the network breakdown, and only incurs 15.6% more controllers than the optimal algorithm. While, the computation time of offline algorithms and online algorithms is just one in a thousand of the optimal algorithm.

The rest of this paper is structured as follows. Section 2 reviews background. Section 3 presents the model and problem formulation. The design of offline algorithm is presented in Section 4. We introduce the ESDN in Section 5, and present the online algorithms in Section 6. The evaluation of the architecture is presented in Section 7. Section 8 summarizes the related work and Section 9 concludes this paper.

2. Background

Software Defined Networking (SDN). SDN decouples the network control plane and forwarding plane enabling (i) the network to become easily programmable, and (ii) the underlying infrastructure to be abstracted for network controller and applications. Network intelligence is centralized in controllers that maintain a global view of the network of software switches, commodity hardware and hosts, which are dumb forwarding entities.

OpenFlow. The OpenFlow protocol [11,12] defines the distributed SDN architecture and commands that enable the interaction between the controllers and switches. In the distributed SDN network, each switch can attach to more than one controller. Only one of the controllers can act as the master controller, while others act as peer or slave controllers. The role of the master controller and the slave controllers can be changed. When the controller role changes, the slave controller first changes its role to a peer controller, then the master controller changes its role to a slave controller, at the same time, the peer controller acts the master role. These role changes are realized by Role-Request message and Role-Replay message.

Even though the OpenFlow protocol defines the role change process, it cannot ensure the safety and liveness properties of the network [10]. To tackle these issues, Dixit et al. [10] proposed an elastic distributed SDN controller system, which presents a switch migration protocol. In the protocol, switches that controlled by a controller can be seamlessly migrated to another. However, the design does not discuss how to achieve the architecture adaptive to runtime network load, and only includes a basic elastic SDN controller architecture and a switch migration proposal.

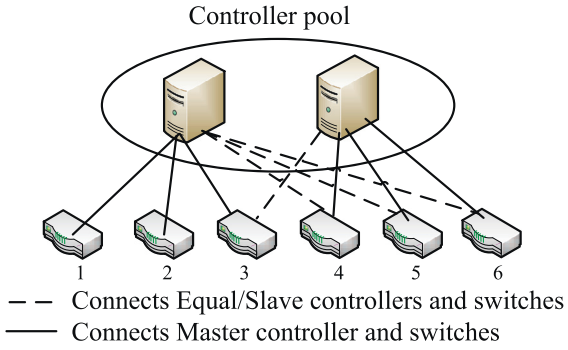

Fig. 1. An illustration of the network.

Fig. 1 shows an example of a network topology. The solid line connects switches and a master controller, and the dotted line connects switches and an peer or slave controller. Two controllers are selected and active in the controller pool now. The selected controllers will be changed with the change of network load. For example, when the load of one controller falls below a given threshold and meets some conditions (see Section 3), it can be inactivated. Note that, since both peer and slave controllers do not respond to controller messages, e.g., Packet-In packet, for simplicity, we do not distinguish them in this paper.

3. Problem statement and modeling

We present the controller selection problem in this section. In a network, each SDN switch (switch for short) has one or more connections to the controllers. Each switch has a traffic demand, which reflects the traffic volume (control and configuration messages, etc.) between the switch and the controller. On the one hand, a controller cannot hold a traffic volume that exceeds the capacity. More strictly, the traffic volume should not exceed a threshold to improve the resilience against a sudden traffic increasing or switch migrations. A controller whose traffic volume is zero can be inactivated. Our target is to minimize the number of active controllers, and achieve load balancing. The model can be applied to the scenarios that the network load increases. In these scenarios, switches can migrate and re-attach to controllers if controller load exceeds the threshold.

Formally, let $\mathcal{C}$ denote the set of controllers and $|\mathcal{C}| = n$. For each controller $c_i \in \mathcal{C}$ $(1 \leq i \leq n)$, let T_i be the capacity, i.e., the maximum traffic volume that c_i can hold, and α $(0 < \alpha \leq 1)$ be a threshold that the ratio of the actual traffic volume to T_i must not exceed. Let N_i be the maximum number of switches that controller c_i can hold. Note that we use N_i to constrain the failure group size and provide better resilience against failures. That is, a single controller failure should not lead to too large a network failure, and the control plane should be restored within a short time. Let B_i be a binary variable that indicates whether controller c_i is active. Let $\mathcal{S}$ denote the set of switches and $|\mathcal{S}| = m$. For each $c_i \in \mathcal{C}$ $(1 \leq i \leq n)$ and $s_j \in \mathcal{S}$ $(1 \leq j \leq m)$, $R_{i,j}$ is a ternary variable that indicates whether there can be a connection between switch s_j and controller c_i. Since there is a need to manage latency and traffic, not all controllers may be able to manage all switches, and the value of $R_{i,j}$ should be set based on latency and bandwidth in practice. In particular, if there can be a connection between the master controller and the switch, $R_{i,j} = 1$; if there can be a connection between the slave/peer controller and the switch, $R_{i,j} = -1$; otherwise $R_{i,j} = 0$. Let F_j be the traffic demand of switch s_j, and $\delta_{i,j}$ be the fraction of traffic demand F_j that is carried by the connection

Please cite this article as: Y. Chen et al., Adaptive Distributed Software Defined Networking, Computer Communications (2016), http://dx.doi.org/10.1016/j.comcom.2016.11.009

Table 1
Notations used in this paper.

Notation	Meaning
α	The threshold that the ratio of the actual traffic volume must not exceed, and $0 < \alpha \leq 1$.
β	The threshold that the ratio of the actual traffic volume to shrink the controller pool, and $0 \leq \beta < 1$.
$\mathcal{C}$	The set of controllers, and $\|\mathcal{C}\| = n$.
$\mathcal{S}$	The set of switches, and $\|\mathcal{S}\| = m$.
τ	Hard time trigger of the Monitor module.
E	Network changes trigger of the Monitor module.
D	Threshold of load deviation to balance the network.
B_i	A binary variable that indicates whether the ith controller is active.
$R_{i,j}$	A ternary variable indicating the possible connection between the jth switch and the ith controller.
F_j	The traffic demand of the jth switch.
T_i	The capacity of ith controller.
N_i	The maximum number of switches that the ith controller can hold.
$\boldsymbol{B}$	The set of B_i.
$\boldsymbol{R}$	The set of $R_{i,j}$.
$\boldsymbol{T}$	The set of T_i.
$\boldsymbol{F}$	The set of F_j.
$\delta_{i,j}$	The fraction of F_j that is carried by the connection between the ith controller and the jth switch.

between c_i and s_j. Table 1 summarized the key notations used in the paper. We model the controller selection problem as follows.

$$\min \sum_{i=1}^{n} B_i \tag{1}$$

subject to: $$B_i \in \{0, 1\}, \qquad \forall i \in \{1, \ldots, n\} \tag{2}$$

$$\sum_{i=1}^{n} \delta_{i,j} B_i = 1, \qquad \forall j \in \{1, \ldots, m\} \tag{3}$$

$$\delta_{i,j} \in [0, 1], \quad \forall i \in \{1, \ldots, n\}, \quad \forall j \in \{1, \ldots, m\} \tag{4}$$

$$\sum_{i=1}^{n} \delta_{i,j} \mid R_{i,j} \mid = 1, \qquad \forall j \in \{1, \ldots, m\} \tag{5}$$

$$\sum_{j=1}^{m} \delta_{i,j} \mid R_{i,j} \mid F_j \leq \alpha T_i. \quad \forall i \in \{1, \ldots, n\} \tag{6}$$

$$\sum_{j=1}^{m} \delta_{i,j} \mid R_{i,j} \mid \leq N_i. \qquad \forall i \in \{1, \ldots, n\} \tag{7}$$

Note that, Eq. (1) is our objective, i.e., minimizing the number of active controllers. Eq. (2) means that each controller is either active or inactive. Eq. (3) means that the traffic demand of each switch must be fully satisfied by active controllers. Eq. (4) means that the traffic demand of each switch must be delivered as a whole. Eq. (5) means that the traffic demand of each switch must be satisfied through only one existing connection.[1] Eq. (6) means that the total traffic volume on a controller must be less than a given threshold. Eq. (7) means that the number of switches held by each controller should be less than the maximum. The input variables of the model are m, n, $R_{i,j}$, F_j, α, and T_i, while the decision variables are B_i and $\delta_{i,j}$. Now we analyze the complexity of the problem.

Theorem 1. *The controller selection problem is NP-Hard.*

Proof. We prove the theorem by a polynomial time reduction from the bin packing problem, which is known to be NP-hard [13], to the controller selection problem.

The bin packing problem is to pack a finite number of objects with weights $w_1, \ldots, w_m$ into a finite number of bins that have the same capacity W, in a way that minimizes the number of bins used. For each instance of the bin packing problem, we can construct an instance of the controller selection problem in polynomial time as follows. For each object with weight w_j, add switch s_j into $\mathcal{S}$, and set the traffic demand to w_j, i.e., $F_j = w_j$. Then, construct a set of enough number of controllers, where each controller c_i has the same capacity $T_i = W$. Let α be 1. Let $R_{i,j} = 1$ for each (i, j) pair, such that the traffic demand of any switch can be satisfied by any controller. Note that in our constructed instance of the controller selection problem, all controllers have the same capacity, and some parameters are assigned special values. However, if the constructed sub-problem is equivalent to a known NP-Hard problem (i.e., the bin-packing problem in our case), then the whole problem is NP-Hard. In the constructed problem, the controllers can be seen as the bins while the switches are corresponding to the objects, and we can see that the optimal solution to the bin packing problem is also the optimal solution to the constructed problem. Thus, the two problems are equivalent, and this ends our proof. □

4. The offline algorithm

In this section, we develop an offline algorithm to solve the controller selection problem. Term "offline" here means that, each time the traffic demand changes, the algorithm selects and allocates one appropriate controller for *every* switch. That is, the algorithm does not take into account the previous connection between each switch and each controller. Instead, the connections are constructed from the very beginning. Such an operation pattern seems not practical. In a large network with thousands of switches, one would expect to adjust only a few connections when necessary. However, the offline solution has some theoretical importance and can give us in-depth insights to the problem. First, we will present an approximation ratio to the problem, which implies the tradeoff between stability and flexibility that can be made by the online algorithm at current stage. Further, some basic principles can be obtained to develop the online algorithm, so as to achieve a similar approximation ratio.

In each round of the offline algorithm, a controller is selected as the master controller of a given switch. The selection principles are as follows: 1) the controller selected must be one of the controllers that can have a connection with the switch; 2) the controller should have the most traffic among all controllers that can be a master controller; and 3) the utilization ratio of the selected

[1] Note that an existing connection may not connect to an active controller, so we need both Eqs. (3) and (5).

controller must not exceed αT_i after adding the traffic of current switch under consideration.

Algorithm 1 The Offline Algorithm

Input: $\boldsymbol{R}$, $\boldsymbol{T}$, $\boldsymbol{F}$, α
Output: $\boldsymbol{B}$, δ_{ij} for each (i, j) pair

1: $M_i \leftarrow 0$ for each i;$\delta_{ij} \leftarrow 0$ for each (i, j) pair;
2: **for** $j = 1 \rightarrow m$ **do**
3: $Max\_F = 0$; $\hat{i} = 0$;
4: **for** $i = 1 \rightarrow n$ $(R_{ij} \neq 0)$ **do**
5: **if** $(M_i + F_j > Max\_F)$ and $(M_i + F_j \leq \alpha T_i)$ **then**
6: $Max\_F \leftarrow M_i + F_j$; $\hat{i} \leftarrow i$;
7: **end if**
8: **end for**
9: $\delta_{\hat{i}j} \leftarrow 1$;$M_{\hat{i}} \leftarrow M_{\hat{i}} + F_j$;
10: **end for**
11: **for** $i = 1 \rightarrow n$ **do**
12: **if** $M_i = 0$ **then** $B_i \leftarrow 0$;
13: **else** $B_i \leftarrow 1$;
14: **end if**
15: **end for**
16: **return** $\boldsymbol{B}$, δ_{ij} for each (i, j) pair;

The pseudo-code of the offline algorithms is shown in Algorithm 1. The inputs are $\boldsymbol{R}$, $\boldsymbol{T}$, $\boldsymbol{F}$ and α. The outputs are $\boldsymbol{B}$ and δ_{ij} for each (i, j) pair. Step 1 initializes M_i and δ_{ij} to 0, where M_i is used to record the current total traffic amount of the ith controller. In each round of Steps 2–10, the algorithm selects one controller for a switch. MAX_F in Step 3 records the maximum traffic amount of the controllers at current stage, and $\hat{i}$ records the corresponding controller, which will be selected in Step 9. The first selection principle discussed above is realized by Step 4, which only checks the controllers that can have a connection to the switch. The second and the third selection principles are realized by the conditions in Step 5. Note that after a controller is selected, the traffic amount of the controller, i.e. $M_{\hat{i}}$, is updated in Step 9. After selecting a controller for each switch, Steps 11-15 compute $\boldsymbol{B}$ based on the total traffic amount of each controller. A controller without any traffic can be inactivated as shown in Step 12.

Theorem 2. *The computational complexity of the offline algorithm is $O(mn)$ in the worst case, where n and m are the number of the controller and switch nodes, respectively.*

Proof. We prove the theorem in two steps. First, we show the computation complexity for a fully connected network, in which each switch is connected to all the controllers. In this network, one switch can be migrated to any controller when the traffic is small enough. The offline algorithm allots one controller for one switch each time. For each switch, the algorithm should pick one controller from all the controllers (the number of the controllers is n), and there are m switches in total, so the computational complexity is $O(mn)$.

The second step of our proof is to show that the computation complexity is less than that in the above case. In particular, in a network that is not fully connected, the number of controllers each switch connected is less than n. Then, following the offline algorithm, the computation complexity is less than $O(mn)$. This ends our proof. □

Theorem 3. *In our model, in which every switch is connected to one master controller and at most one slave controller, the offline algorithm is a 2-approximation heuristic.*

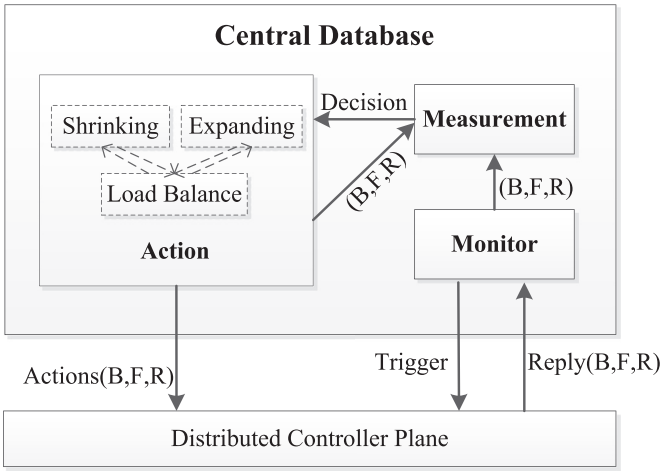

Fig. 2. ESDN design.

Proof. We have already shown that the offline algorithm runs in polynomial time. To show that the offline algorithm is a 2-approximation heuristic, we assign a cost of 1 to each controller selected by the Optimal algorithm, distribute this cost to one switch whose master controller is the controller, and set the cost of other switches whose master controller is the controller to 0. The number of the active controllers of Optimal solution ζ^* is

$$|\zeta^*| = \sum_{s \in S} cost_s; \tag{8}$$

Let S_i denote all the switches connected to the ith controller c_i, ζ denote the solution of the offline algorithm, then we have

$$|\zeta^*| \leq \sum_{c_i \in \zeta} \sum_{s \in S_i} cost_s = |\zeta| \sum_{s \in S_i} cost_s; \tag{9}$$

Because one switch is connected to at most two controllers, then we have

$$|\zeta| \sum_{s \in S_i} cost_s \leq 2 \sum_{s \in S} cost_s = 2|\zeta^*|; \tag{10}$$

The $\sum_{s \in S_i} cost_s$ is an integer that not less than 0, then

$$|\zeta| \leq 2|\zeta^*|; \tag{11}$$

□

5. ESDN design

The traffic amount in a real-world network is changing quickly. The offline algorithm recomputes the entire network configurations, which is unstable. Thus, we need a practical design to take place of the offline approach. In particular, the active controllers should be selected based on the previous status of the network, and only a part of the controllers and the switches should change so as to adapt to network flow changes [10]. In this section, we propose an adaptive elastic distributed SDN architecture (ESDN) which enables dynamic controller selection and switch migration by online algorithms adapting to network traffic changes.

5.1. Overview

In ESDN, the network load is monitored by a central database, which includes three main modules, Monitor module, Measurement module and Action module (see Fig. 2). The Monitor module monitors the network load all the time, and checks the network load statistical information when triggered. Then the Monitor

Please cite this article as: Y. Chen et al., Adaptive Distributed Software Defined Networking, Computer Communications (2016), http://dx.doi.org/10.1016/j.comcom.2016.11.009

module transmits the reply information from the distributed controller plane to the Measurement module. The Measurement module makes decisions to adjust the network. Then the Action module takes actions according to the decisions from the Measurement module. The results of the actions are returned to the Measurement module to make further decisions. There is an iteration process between the Measurement module and Action module, which is to finish when the decision is Null.

5.2. Monitoring the network

The Monitor module is to gain statistical information of the network. It monitors the network all the time, and requires the network information when triggered. There are two methods to trigger the Monitor module, hard time off trigger and network changes trigger. Hard time means the period time τ to check the network, which is set by the operator. While the network changes means that the network changes a lot, which exceeds the threshold E set according to the network requirement. The period time τ and network changes threshold E can be set according to network time sensitivity. When the network topology changes quickly or the network flow varies largely, τ and E should be set to a small value. When the network is much stable, large τ and E can be set. After getting the information, the Monitor module returns the information to the Measurement module.

5.3. Making decisions and adjusting the network

The measurement module makes decisions to adjust the network. When the load ratio of some controllers exceeds a given threshold, the controller pool is expanded by activating more controllers, to prevent the network from breaking down. When the load ratio of some controllers falls below a given threshold, the controller pool is shrunk by inactivating these controllers to save power. We note that uneven controller loads may incur an overload or a small mean load, but the controller pool may not need change in such cases. To avoid unnecessary controller reselection, we need to balance the controller loads when the loads are unevenly distributed. Load balance is also needed when the network load is uneven after changing the controller pool and migrating the switches.

Specifically, the load of each controller is measured, and different decisions are made under the following conditions. First, we compute the deviation of the controller loads. If the deviation is greater than a given threshold, the network load is uneven and load balance should be performed. Second, we compute the controller utilization ratio as the ratio of the load to the controller capacity. If the utilization ratio of all controllers is greater than α, the controller pool should be expanded. Third, if the minimum controller utilization ratio is less than β, the pool should be shrunk.

The Action module is used to take actions to adjust the network, according to the decision made by the Measurement module. The actions, i.e., load balance, expanding the controller pool, and shrinking the controller pool, will be discussed in detail in the next section. Only one action will be taken each time. After the network is adjusted, the traffic load information changes accordingly, which is captured by the Measurement module to make further decisions.

6. The online algorithms

Based on the information of the network, we present the online algorithms of the three actions. According to the upper threshold α and lower threshold β of the controller load ratio, the online algorithms only adjust a small part controllers and switches.

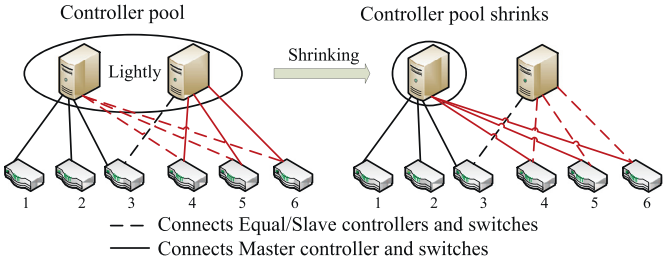

Fig. 3. Shrinking controller pool.

6.1. Shrinking controller pool

The controller pool needs to shrink when the shrinking action is called. First, the algorithm locates all the controllers that the load ratio below the threshold β. Then, the algorithm reconnects all the switches attached to the controllers. If one controller does not have any attached switches, it goes to an inactive state. Unlike the polynomial time algorithms, the online algorithms do not need to create a virtual network, instead, the online algorithms adjust the network on the real network directly according to the current information of the network.

To find the controller that can be inactivated, we visit the least-load controller that the load ratio below β, and migrate the switches attached to the controller to their most-load slave controllers. If all switches attached to the controller are migrated, it goes to an inactive state. Fig. 3 shows an example of shrinking the controller pool, the left part indicates the network before shrinking, in which there are two active lightly-loaded controllers in the controller pool. The right part shows the controller pool after shrinking. The third, the fourth and the fifth switches are migrated to the left controller, and the algorithm inactivates the right controller.

Algorithm 2 Shrinking controller pool of online algorithms

Input: $\boldsymbol{B}$, $\boldsymbol{R}$, $\boldsymbol{T}$, $\boldsymbol{F}$, α
Output: $\boldsymbol{B}$, $\boldsymbol{R}$, $\boldsymbol{F}$

1: $S_s[]\_Ascending(M)$;
2: **for** $(j \in S_s[])$ **do**
3: $M_i = \sum_{j=1}^{m} \delta_{i,j} \mid \boldsymbol{R}_{i,j} \mid \mathbf{F}_j$;
4: $Max\_F = 0$; $sl = 0$; $ma = 0$;
5: **for** $(i \in M\_S[j])$ **do**
6: **if** $((M_i + F_j > Max\_F)$ && $(M_i + F_j \leq \alpha \boldsymbol{T}_i))$ **then**
7: $Max\_F = M_i + F_j$; $la = i$;
8: **end if**
9: **if** $(B_i == 1)$ **then**
10: $ma = i$;
11: **end if**
12: **end for**
13: $Connect(C_{sl}, S_j)$; $R_{sl,j} = 1$; $R_{ma,j} = -1$;
14: **end for**
15: $Update(\boldsymbol{B}, \boldsymbol{R}, \boldsymbol{F})$;
16: $return(\boldsymbol{B}, \boldsymbol{R}, \boldsymbol{F})$;

Algorithm 2 shows the pseudo-code of the shrinking action of online algorithms. The inputs include $\boldsymbol{B}$, $\boldsymbol{R}$, $\boldsymbol{T}$, $\boldsymbol{F}$ and α, and the outputs include $\boldsymbol{B}$, $\boldsymbol{R}$ and $\boldsymbol{F}$. First, the algorithm computes all the switch connected to the controllers that load ratio below β, the number of which are stored in $S_s[\,]$, which is sorted in ascending order (Step 1). Then, as the shrinking action of the polynomial time algorithm, the algorithm computes the computes the total load amount of each controller c_i of the network, and initializes the load Max_F and the number sl of the controller that has the

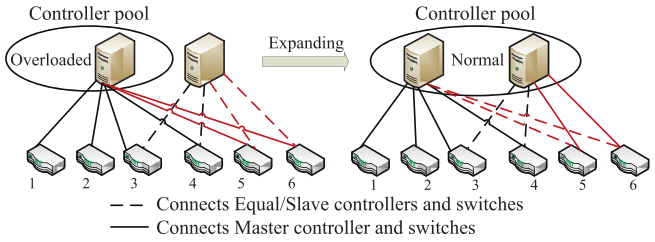

Fig. 4. Expanding controller pool.

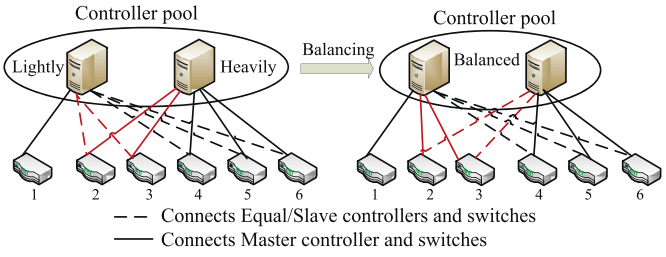

Fig. 5. Balancing load among controllers.

most load after connecting the switch. More, the algorithm records the current master controller *ma* of the switch to update the network after the migration process (Steps 4, 9, 10, 15). When one controller does not have switches attached to it, the update function changes its status (Step 15).

6.2. Expanding controller pool

The controller pool is expanded by activating more controllers to prevent the network from breaking down when the load ratio of some controllers exceeds a given threshold α. To expand the controller pool, the algorithm migrates the switches from the heavily-load controller to one inactivated controller. As the shrinking action, the expanding action also adjusts part controllers not the whole network. Fig. 4 presents an example of expanding the controller pool, the left part indicates the network before expanding, in which there is just one active controller in the controller pool, which is overloaded. The right part shows the controller pool after expanding. The right controller has been activated, and the last two switches are migrated to the left controller.

The algorithm of the expanding action is much similar as the shrinking action, except the adjusting conditions. Instead computing the $S_s[\,]$, the algorithm computes the $S_e[\,]$, which records all the switches attached to the controllers that the load ratio exceeds the threshold α. Other processes are as in the Algorithm 2.

6.3. Balancing load among controllers

As discussed above, the network is dynamically changing and the loads of different controllers may be uneven. Thus, we need to balance the controller loads to improve network performance and avoid controller crash. The load balance action of the online algorithm is much different from the polynomial time algorithm. When balancing the network, the algorithm computes the deviation of two controllers before and after the migration to decide whether the connection is appropriate. For reducing the network changes, we just consider the case, in which the load ratio of some controllers exceeds α.

We visit the most-load controller, and check whether a migration is feasible for switches attached to the controller. Fig. 5 shows an example of such migration. The left part indicates the network before load balancing, and the right part indicates the network after load balancing. Before load balancing, only one switch attaches to the left controller, which is lightly loaded, and five switches attach to right controller, which is heavily loaded. In order to balance the network, we migrate the second and the third switches from their master controller (the right one) to their slave controller (the left one), and the loads are balanced.

Algorithm 3 Balancing load among controllers

Input: $\boldsymbol{B}$, $\boldsymbol{R}$, $\boldsymbol{T}$, $\boldsymbol{F}$, α
Output: $\boldsymbol{B}$, $\boldsymbol{R}$, $\boldsymbol{F}$

1: $M_i = \sum_{j=1}^{m} \delta_{i,j} \mid \boldsymbol{R}_{i,j} \mid \boldsymbol{F}_j$; $(i \in (1, ..., n))$
2: $D_{x,y} = \mid M_x - M_y \mid$;
3: $S_e[\,]\_Descending(M)$;
4: **for** $(j \in S_e[\,])$ **do**
5: $Min\_D = \infty$; $sl = 0$;
6: **for** $(i \in Sl[j])$ **do**
7: **if** $((D_{Ma[j],i) < Min\_D)$ && $(M_i + F_j \leq \alpha \boldsymbol{T}_i))$ **then**
8: $Min\_D = D_{Ma[j],i}$; $sl = i$;
9: **end if**
10: **end for**
11: $Connect(C_{sl}, S_j)$; $R_{sl,j} = 1$; $R_{Ma[j],j} = -1$;
12: **end for**
13: $Update(\boldsymbol{B}, \boldsymbol{R}, \boldsymbol{F})$;
14: $return(\boldsymbol{B}, \boldsymbol{R}, \boldsymbol{F})$;

Algorithm 3 shows the load balance action of the online algorithms. The inputs include $\boldsymbol{B}$, $\boldsymbol{R}$, $\boldsymbol{T}$, $\boldsymbol{F}$ and α, and the outputs are $\boldsymbol{B}$, $\boldsymbol{R}$ and $\boldsymbol{F}$. We use $D_{x,y}$ to indicate the load difference value between controller x and controller y (Step 2). Then the algorithm sorts the $S_e[\,]$ (the same meaning as the expanding action) in descending order (Step 3). For each switch in the $S_e[\,]$, the algorithm defines the minimum deviation *Min_D* during the process and the number of the controller *sl* that the switch would migrated to (Step 5). For each slave controllers of this switch in $C_{Sl[j]}$, if the deviation between the slave controller and the current master controller of the switch $C_{Ma[j]}$ is smaller than *Min_D*, and the load after adding the flow of the switch does not exceed αT, the algorithm records the slave controller, and updates *Min_D* (Steps 7-9). Then the algorithm updates the network (Steps 11, 13).

Computational complexity. Now we analyze the computational complexity of online algorithms. The online algorithms adjusts part network instead of the whole network as in the polynomial time algorithms. And the online algorithms preprocess the information of the network. making the process taken in some order. The worst case is that the load ratio of every controller exceeds the upper threshold α or falls below the lower threshold β, in which the computational complexity of the online algorithms is $O(mn)$. In general cases, just a small part of the switches should be migrated, in which the computational complexity of the online algorithms is much lower $O(mn)$.

7. Evaluation

7.1. Methodology

For the simulations, since known public data of SDN networks consist of too small topologies (for example, the Stanford backbone has two controllers and fourteen switches), we generate the topologies randomly, which have similar size as real networks obtained from [14]. Specifically, we generate the topologies to ensure that each switch attaches to only one master controller. Then, we generate slave controllers randomly for each switch and guarantee that the number of slave controllers is less than $n-1$. Table 2 summaries some topologies used in the simulations. For

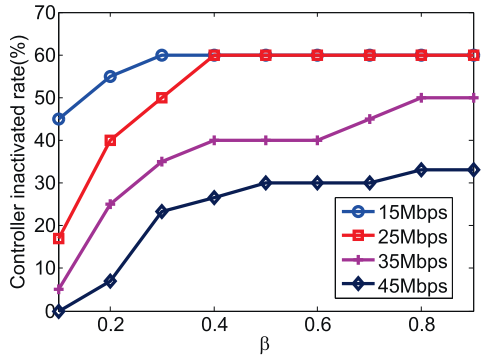

Fig. 6. The effectiveness of β to the controller inactivated rate when in different network load.

Table 2
Network of different size.

Scene NO.	Controller number	Switch number
1	5	50
2	7	50
3	9	50
4	11	50
5	11	100
6	11	150
7	11	200

each scene, the results are the average values over thirty independent simulations.

The traffic matrixes used for simulations are generated randomly based on the synthetic topologies. The total flow of each controller is not larger than αT. For simplicity and clarity of the results, we set the capability of all controllers as the same, which is 100 Mbps. The results are similar for a larger controller capability. The simulations are performed on a machine with an Intel Core i5 3470 CPU at 3.2 GHz, and 1GB of RAM, running 64bit Windows 7.

The key parameters of our algorithms, i.e., α and β, are set as follows. First, α is the threshold of controller utilization ratio to perform expanding of the controller pool. In the Internet, a link utilization ratio of 0.5 is often used as threshold for traffic engineering to avoid congestion, but there is no similar data for a controller utilization ratio. In our simulations, we set α to 0.8, because controllers are supposed to have a large capacity, and should be able to handle traffic burst along with the load balance, when the utilization ratio is less than 0.8.

Second, for β - the lower threshold of controller utilization ratio to perform shrinking of active controllers, we simulate the online algorithms to show how β can impact the performance. The controller number is set to 10, and the switch number is set to 20. Fig. 6 shows the results, where each curve shows the rate of the inactivated controllers with certain average network load. We can see that the number of the controllers that can be inactivated increases with β. However, the increment is little when β is large. For instance, the inactivated controllers remain as 60% when β is greater than 0.3 in the 15 Mbps scene. This implies that some controller in the network is always light-weighted, and the active controller pool can be shrunk even when the lower threshold is large. Based on the results, we set β to 0.4 in our simulations. In the mean time, the average network load is set to 25 Mbps.

We evaluate both the offline and the online algorithms. For comparison, we also show the optimal solution, which is obtained by enumerating all feasible solutions. We do not evaluate the approaches proposed by existing studies, because our problem is different from other load balance problems. Existing studies, e.g. [10], do not consider the optimization of dynamic switch migration with traffic load. Instead, our work focuses on how to dynamically shrink and expand the controller pool, as well as switch migrations and network load balance.

Moreover, we conducted Mininet experiments, and simulated multi-controller in Floodlight in the experiments. We built an out-of-band control channels using network virtualization technologies (e.g., Linux Namespace, Veth Piar and Linux Bridge) so as to measure the load of different controllers and demonstrate the effectiveness of our algorithms in Mininet. Due to limits of computation and memory resources in the machine, we generate a small full mesh topology with four switches. Each switch with two attached hosts connects four controllers with the out-of-band control channel. Each host generate new packets so as to trigger packet-in messages in switches. Our experiments are composed of two phases. We generate a small number of packet-in messages in the first phase to shrink active controllers, and produce a large number of packet-in messages in the second phase to expand the active controller. For simplicity, the maximum load of each controller is set be 3.75 Mbps.

7.2. Results

7.2.1. Algorithm effectiveness

Fig. 7(a) shows the inactivated controller number when the traffic amount decreases. We can see that the inactivated controller number is approximatively proportional to the total controller number in a network, and this implies that our algorithms can dynamically shrink the active controller pool with different network sizes. The inactivated controller of the online algorithms is less than the offline algorithm, but the results are very close. This is because our online algorithms follow the basic principles explored by the offline analysis.

Fig. 7(b) shows the reactivated controller number when the traffic amount increases. The number of active controllers before expanding is 5, and we can see that the reactivated controller

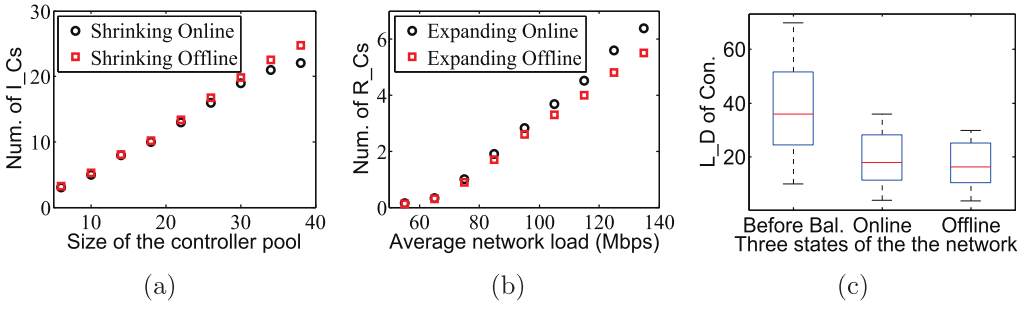

Fig. 7. The effectiveness of three actions of Offline and Online algorithms. (a) Shrinking action (I_Cs is short for inactivated controllers); (b) Expanding action (R_Cs is short for reactivated controllers); (c) Load balancing action (L_D is short for load deviation).

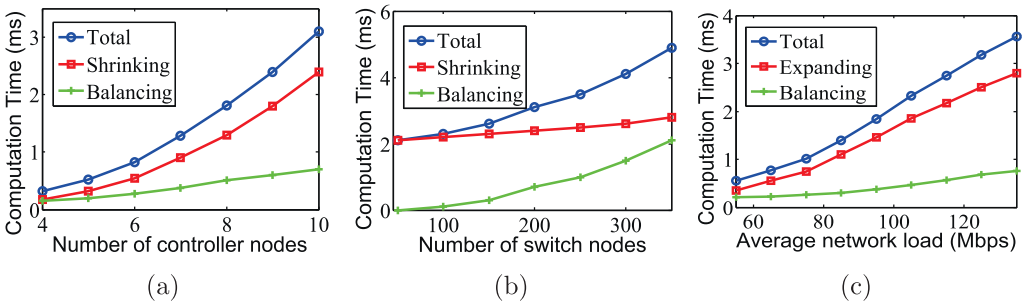

Fig. 8. The computation overhead of three actions of offline and online algorithms. (a) Controller nodes and computation time of shrinking action and load balancing action; (b) Switch nodes and computation time of shrinking action and load balancing action; (c) Controller nodes and computation time of expanding action and load balancing action.

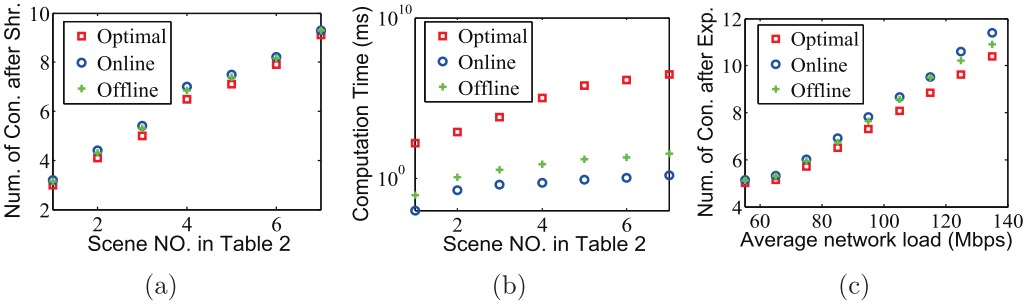

Fig. 9. Comparison with optimal algorithm. (a) Controller number after shrinking by three algorithms (Shr. is short for shrinking action); (b) Shrinking computation time of three algorithms; (c) Controller number expanded by three algorithms in different network load (Exp. is short for expanding action).

number is approximatively proportional to the traffic amount. For example, the number of active controllers is doubled when the traffic amount is doubled. This is not surprising because our algorithms select controllers and perform switch migration to adapt the traffic amount dynamically. Again, the results of the offline and the online algorithms have a similar relation to that in Fig. 7(a), which means that the online algorithm can still solve the controller selection problem effectively while the flexibility is improved.

Fig. 7(c) shows the deviation of controller loads. We can see that after applying the load balancing action, the deviation is reduced by about 50%, for all the maximum, the average, and the minimum. The result of the offline algorithm is a little better than the online algorithm. This is because the offline algorithm performs load balancing along with the controller selection, while in the ESDN system, controller selection and load balance are separate modules. And this implies that, simply balancing the load (as in existing studies) cannot achieve a load balance as good as a load balance with dynamic adjustment of the controller pool.

7.2.2. Computation overhead

Fig. 8(a) shows the computation time of the shrinking and the load balancing algorithms as a function of controller number. We can see that the algorithms to take the actions have little overhead, i.e. a few milliseconds even when the controller number increases. Such a time is acceptable for responding to load changes. Further, the computation time of the shrinking algorithm increases super-linearly with the controller number. This is because the computation complexity of the shrinking algorithm is $O(mn)$ in the worst case, and when the controller number increases, the switch number also increases in our simulations.

Fig. 8(b) shows the computation time of the shrinking and the load balancing algorithms as a function of switch numbers. The computation time is similar to that in Fig. 8(a), i.e., a few milliseconds. However, we see that the computation time of the shrinking algorithm changes little with the switch number, while the computation time of load balancing increases quickly. This is implies that, though the computation complexity of the shrinking algorithm is $O(mn)$ in the worst case, the computation overhead is less in general, especially with respect to the switch number. On the other hand, the load balancing algorithm needs more computations when there are more switches.

Fig. 8(c) shows the computation time of the expanding and the load balancing algorithms as a function of traffic amount. The initial number of active controllers is 5. We see that when the load increases, the computation time needed also increases. This is because more controllers are activated to handle the traffic, and recall that the computation complexity in the worst case is $O(mn)$. However, the total computation time is increasing sub-linearly when the traffic amount is large, and is less than 4 ms when the average network load reaches 135 Mbps.

7.2.3. Comparison with optimal algorithm

Fig. 9(a) shows the number of active controllers with increasing network size. We can see that the results of our algorithms are very close to that of the optimal solution. This means that our algorithms can find a near-optimal solution in a short time. The active controller number increases linearly, similar to the results in Fig. 7(a).

Fig. 9(b) shows the computation time of the algorithms. Note that the y-axis is log-scaled, and we can see that our algorithms have much less computation time compared to the optimal solution. The online algorithm needs less time than the offline algorithm. This means that our algorithms can find the near-optimal solutions efficiently. For instance, when the number of controllers or switches increases, the time consumed by the optimal algorithm is several thousand times of that consumed by our algorithms.

Fig. 9(c) shows the number of active controllers as a function of traffic amount. Again, we see that the optimal solution has the minimum number of active controllers, and the results of our algorithms are a little higher. However, the results are very close and

Please cite this article as: Y. Chen et al., Adaptive Distributed Software Defined Networking, Computer Communications (2016), http://dx.doi.org/10.1016/j.comcom.2016.11.009

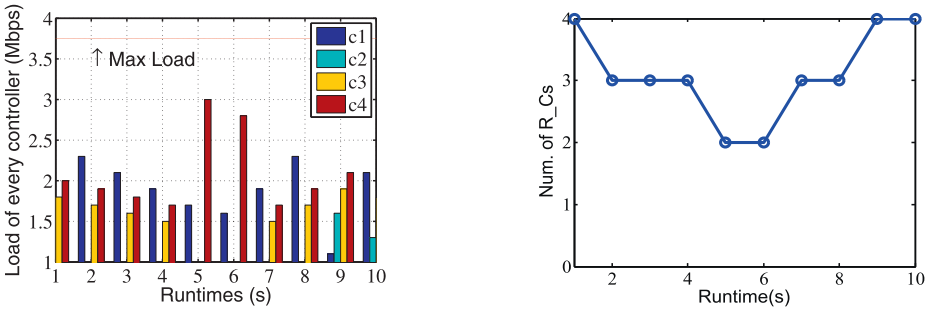

Fig. 10. Experiment in Mininet. (a) Network load of every controller; (b) Number of activated controllers(R_Cs is short for reactivated controllers).

the online algorithm only incurs 15.6% more controllers compared to the optimal algorithm in the worst case.

7.2.4. Experiment in Mininet

Fig. 10(a) shows the network load of every controller in the Mininet experiment, and Fig. 10(b) shows the number of activated controller in the network. Initially, the load in controllers, i.e., c1, c2, c3, c4, are set to be 1, 1.5, 1.8, and 2.0 Mbps, respectively. As the number of packets-in messages generated by each switch decreases and the whole network load decreases, the shrinking action will be activated. As shown in Fig. 10(a), the average network load in c2 and c3 reduce in the first phase (within 6 s). The load of these two controllers decline to 0 Mbps, and these two controllers are inactivated. Meanwhile, since the packet-in messages are shifted to the other controllers, i.e., c1 and c4, their average network load reaches 1.7 and 3 Mbps, respectively. Therefore, the results demonstrate that the shrinking action can effectively shrink the network effectively.

In the second phase, i.e., starting from 7 s (see Fig. 10(a)), the traffic in switches increase, and the whole network load increases. Thereby, the expanding action will be activated. We can observe that the average network load increases from 0 to 2 Mbps. Controllers c2 and c3 are activated, and the average network loads in c2 and c3 increase to 1.9 and 2.1 Mbps, respectively, at the ninth second. Meanwhile, the average network loads in c1 and c4 decreases to 1.1 and 2.1 Mbps, respectively. As shown in Fig. 10(b), with the increase in the network load, the number of activated controllers increases from two to four.

8. Related work

The SDN controller architecture has been developed from a centralized one to a distributed one. The centralized system experienced single-threaded design [15] and multi-threaded design [16–18] in recent years. For instance, Yeganeh et al. [17] proposed Kandoo which used two layers of controllers to control the whole network. The bottom layer is a set of controllers disconnected to each other, and the top layer is the logically centralized controller. The design of Kandoo is much similar to the distributed system.

However, the centralized controller has an issue of scalability and reliability. Thus, multiple distributed SDN controllers [6,7,19–21] are proposed. For example, Onix [7] is a control platform which offers a general-purpose API for control plane implementations. Open Network Operating System (ONOS) [19] presented an experimental distributed SDN control platform. ONOS adopts a distributed architecture for high availability and scale-out. ONOS followed in the footsteps of previous closed source distributed SDN controllers such as [7]. Yanc [20] proposed a controller platform for software-defined networks that exposes network configurations and state as a file system. It enables a network operating system that can be used in various ways so that it can leverage different innovation in the operating system area. HotSwap [21] is a hypervisor that sits between the controller and the switches and provides seamless upgrades for SDN controllers. In these architectures, the mapping between a switch and a controller is statically configured, which cannot well handle bursts and thereby may result in controller crash.

To tackle these issues, Dixit et al. [10] proposed an elastic distributed SDN controller system, ElastiCon, which attracts much attention. With a switch migration protocol, switches that controlled by a controller can be seamlessly migrated to another. However, the design does not discuss how to achieve the architecture adaptive to runtime network load, and only includes a basic elastic SDN controller architecture and a switch migration proposal. They did not develop load adaptation algorithms that compute optimal switch migration strategy respect to dynamic traffic load. but our work focuses on how to dynamically shrink and expand the controller pool with respect to the network load and switch migrations.

To achieve load balance between controllers, Guo et al. [22] proposed Load Variance-based Synchronization (LVS) by eliminating forwarding loops and to lower synchronization frequency in the multi-controller multi-domain SDN network. However, they does not propose some specific mechanisms to implement migrations between controllers and switches according to network load so as to achieve load balance in runtime. In particular, these existing schemes might make load balance complicated under switch migrations and incur a high overhead. However, our elastic controller scheme dynamically shrinks and expands the controller pool with respect to the network load.

In this paper, we develop offline algorithms and online algorithms of polynomial time to solve this controller selection problem. The network controller pool can be dynamically shrank or expanded according to the network changes. When the network load is uneven, the algorithm can also balance the network. We extend out previous work appeared in [23]. In particular, we study the controller selection problem in depth and present a two-step development of the offline and online algorithms. We develop the offline algorithm which is proved to be a 2-approximation heuristic; and we improve the online algorithm to decrease the computational complexity in the worst case form $O(n^2m)$ [23] to $O(mn)$. We discuss the ESDN architecture in detail, which includes monitor, measurements, and action modules. We also strengthen the performance evaluation by experiments based on Mininet.

Please cite this article as: Y. Chen et al., Adaptive Distributed Software Defined Networking, Computer Communications (2016), http://dx.doi.org/10.1016/j.comcom.2016.11.009

9. Conclusion

In this paper, we propose a framework to achieve an adaptive and elastic SDN network, namely ESDN. It dynamically selects a minimum number of active controllers that switches attached to, and changes the mapping between switches and controllers according to the network load. We develop offline algorithms and online algorithms to solve the controller selection problem. Our offline algorithms can compute the best solution, and the online algorithms enable dynamic controller selection with switch migration in runtime. We conduct various experiments to demonstrate the effectiveness of the proposed algorithms.

Acknowledgments

This work was supported in part by the National Key R&D Program of China under grant no. 2016YFB0800102, in part by the National Natural Science Foundation of China under grant no. 61572278 and no. 61502268, in part by the R&D Program of Shenzhen under grant nos. ZDSYS20140509172959989, JSGG20150512162853495, Shenfagai[2015]986.

References

[1] M. Casado, T. Garfinkel, A. Akella, M.J. Freedman, D. Boneh, N. McKeown, S. Shenker, SANE: A Protection Architecture for Enterprise Networks, USENIX, 2006.
[2] M. Casado, M.J. Freedman, J. Pettit, J. Luo, N. McKeown, S. Shenker, in: Ethane: Taking Control of the Enterprise, SIGCOMM, 2007, pp. 1–12, doi:10.1145/1282380.1282382.
[3] A. Greenberg, G. Hjalmtysson, D.A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, H. Zhang, in: A Clean Slate 4D Approach to Network Control and Management, SIGCOMM CCR, 2005, pp. 41–54, doi:10.1145/1096536.1096541.
[4] T.V. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani, T. Woo, The SoftRouter Architecture, HOTNETS, 2004.
[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, in: Openflow: Enabling Innovation in Campus Networks, SIGCOMM, 2008, pp. 69–74, doi:10.1145/1355734.1355746.
[6] D. Levin, A. Wundsam, B. Heller, N. Handigol, A. Feldmann, in: Logically Centralized? State Distribution Trade-offs in Software Defined Networks, HotSDN, 2012, pp. 1–6, doi:10.1145/2342441.2342443.
[7] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, S. Shenker, in: Onix: A Distributed Control Platform for Large-scale Production Networks, OSDI, 2010, pp. 351–364.
[8] S. Schmid, J. Suomela, in: Exploiting Locality in Distributed SDN Control, HotSDN, 2013, pp. 121–126, doi:10.1145/2491185.2491198.
[9] A.A. T. Benson, D. Maltz, in: Network Traffic Characteristics of Data Centers in the Wild, IMC, 2010, pp. 267–280, doi:10.1145/1879141.1879175.
[10] A. Dixit, F. Hao, S. Mukherjee, T.V. Laskshman, R. Kompella, in: Towards an Elastic Distributed SDN Controller, HotSDN, 2013, pp. 7–12, doi:10.1145/2491185.2491193.
[11] OpenFlow White Paper: https://www.opennetworking.org/sdn-resources/-sdn-library/whitepapers.
[12] OpenNetworkFoundation: https://www.opennetworking.org/sdn-resources/onf-specific-ations/openflow.
[13] X. Feng, Computational Complexity, University of Electronic Science and Technology: China Machine Press, 2005.
[14] The internet topology zoo: http://www.topology-zoo.org/index.html.
[15] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, in: Nox: Towards an Operating System for Networks, SIGCOMM CCR, 2008, pp. 105–110, doi:10.1145/1384609.1384625.
[16] Floodlight: http://www.projectfloodlight.org/floodlight/.
[17] S. Yeganeh, Y. Ganjali, in: Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications, HotSDN, 2012, pp. 19–24, doi:10.1145/2342441.2342446.
[18] D. Erickson, in: The Beacon Openflow Controller, HotSDN, 2013, pp. 13–18, doi:10.1145/2491185.2491189.
[19] P. Berde, M. Gerola, Hart, in: Onos: Towards an Open, Distributed SDN OS, HotSDN, 2014, pp. 1–6, doi:10.1145/2620728.2620744.
[20] M. Monaco, O. Michel, E. Keller, in: Applying Operating System Principles to SDN Controller Design, HotNet, 2013, pp. 1–7, doi:10.1145/2535771.2535789.
[21] L. Vanbever, J. Reich, T. Benson, N. Foster, J. Rexford, in: Hotswap: Correct and Efficient Controller Upgrades for Software-Defined Networks, HotSDN, 2013, pp. 133–138, doi:10.1145/2491185.2491194.
[22] Z. Guo, M. Su, Y. Xu, Z. Duan, L. Wang, S. Hui, H. Jonat, Improving the performance of load balancing in software-defined networks through load variance-based synchronization, Comput. Networks 68 (2014) 95–109, doi:10.1016/j.comnet.2013.12.004.
[23] Y. Chen, Q. Li, Y. Yang, Q. Li, Y. Jiang, X. Xiao, in: Towards Adaptive Elastic Distributed Software Defined Networking, IPCCC, 2015, pp. 1–8, doi:10.1109/PCCC.2015.7410280.