DHA: Distributed Decisions on the Switch Migration Toward a Scalable SDN Control Plane

Guozhen Cheng, Hongchang Chen, Zhiming Wang, Shuqiao Chen

National Digital Switching System Engineering & Technological R&D Center

Zhengzhou, China

[Email: guozhencheng@hotmail.com, chc@ndsc.com.cn wangzm05@gmail.com, chenshuqiao1973@163.com]

***Abstract*—Distributed control plane is a promising approach to a scalable software-defined networking (SDN). However, traffic changes could incur load imbalance among individual controllers. Live migration of switches from controllers that are overloaded to those that are underutilized may be a solution to handle peak switch traffic using available control resource. Such migration has to be performed with a well-defined mechanism to fully utilize the available resource of controllers. In this paper, we study a scalable control mechanism to decide which switch and where it should be migrated for a balanced control plane, and we define it as switch migration problem (SMP). The main contributions of this paper are as follows. *First*, we define a SDN model to describe the relation between the controllers and switches from the view of loads. Based on this model, we formulate SMP as a network utility maximization (NUM) problem with the objective of serving more requests under the available control resource. *Second*, we design a synthesizing distributed algorithm for SMP --- distributed hopping algorithm (DHA), by approximating our optimal objective via Log-Sum-Exp function. In such DHA, individual controller performs algorithmic procedure independently. With the solution space $\mathcal{F}$, we prove that the optimal gap caused by approximation is bounded by $\frac{1}{\beta}\log|\mathcal{F}|$, and the DHA procedure is equal to an implementation of a time-reversible markov chain process. *Finally*, the results are corroborated by several numerical simulations.**

Keywords—software-defined networking; scalability; switch migrations; markov chain

I. INTRODUCTION

Scalability is a key issue for the SDN control plane. Distributed control plane is a promising approach to a scalable SDN. Each controller manages a part of switches in the network but makes decisions based on a logically centralized network view. However, the control plane tends to unbalance and its performance will be degraded, since the current static structure between controllers and switches cannot adapt for the network traffic changes.

At first stage, SDN deploys only one central controller that is responsible for all the switches [1]-[3]. This architecture could readily achieve network state consistency and avoid communications among different controllers. But the single resource-limited controller confines the SDN paradigm to a small-scale network, because a large network could experience the overloading of the controller due to the frequent and resource-exhaustive events such as OpenFlow (OF) PACKET-IN events [4]. For example, the flow setup time [1] in an overloaded controller can rise significantly, and its performance can deteriorate rapidly.

A few recent attempts are subsequently taken to tackle this problem via the distributed schemes which may fall into two major groups: those horizontally equalizing all controllers, *i.e.*, flat architecture [4, 5], and those vertically layering from root controller to leaf ones, *i.e.*, hierarchical architecture [6]. Both architectures have comparatively improved the scalability of SDN control plane, but the static map between controllers and switches lead to load imbalance among controllers. For instance, real measurement for network traffic shows that 1-2 orders of magnitude difference between the peak and median flow arrival rates at a switch [7]. Current static configurations could easily induce that some controllers are overcommitted and become hot spots, but other controllers are underutilized and turn to cold spots. Workloads have to be offloaded from the hot spots since there is inadequate resource to meet service level agreements (SLAs). Oppositely, cold spots expect to serve more switches for the high network utility.

[1] Whenever a switch receives a flow, it searches its flow table to find the entry matched. As the match is failed, it requests the controller to calculate the flow path and install appropriate rules. The time required to complete this operation is known as the flow setup time.

ISBN 978-3-901882-68-5 © 2015 IFIP

The recent version of OpenFlow protocol [8] has realized the problem caused by such static configurations. Therefore, it proposes that each switch could be controlled by three different roles of controllers, master, equal and slave. Generally, there is only one master controller for a switch. The master can not only fetch the switches' states but also write rules to its switches to instruct the data plane. The equal controllers are introduced to separate the loads from the master. They have the same authority with master. The slaves only read the states from its switches. Each switch could have more than one equal and slave controllers. If the master is failed due to overload or some exceptions, the equal controllers, or even slaves could be transited to master as soon as possible. However, the OF spec suggests no mechanism explicitly indicating the switch migration or controller roles shift, because the writers of this spec think that this is the responsibility of the controller to choose a master among themselves.

We believe that such migration has to be performed with a well-defined mechanism to fully utilize the available resource of controllers. In this paper, we focus on designing a scalable control mechanism via solving SMP problem. To the best knowledge of the authors, this is the first work to specially solve the SMP for a more balancing SDN control plane. The main contributions in this paper are as follows.

- We first give a SDN model to describe the relation between controllers and switches, and then we define SMP problem as a NUM with the objective of serving more loads under available control resources.
- Based on the Markov approximation framework [14], we approximate our optimal objective with a Log-Sum-Exp function, and design a synthesizing distributed algorithm, distributed hopping algorithms (DHA), to approach the optimal solution of the SMP. We prove that the gap between the approximated solution and the optimal one is limited and the solution search path of our DHA is a markov chain path with a stationary probability distribution.
- We implement a scalable control mechanism based on our DHA algorithms, and validate its performance in real ISP topologies.

The remainder of this paper is organized as follows. The next section gives the related works. The section III discusses the intuition and details of our model for SDN. In section IV, we reduce our SDN model to network utility maximization problem. Section V presents the design of distributed hopping algorithms. Section VI describes an implementation of scalable control mechanism. Section VII validates our DHA. Section VIII concludes this paper.

II. Related Work on Elastic Control

Current static map between controllers and switches prevents controllers from exchanging the network loads according to the traffic changes. To address this problem, ElastiCon [9] is provided to dynamically grow or shrink the amount of controllers set and migrate the switches among controllers along with the network traffic. Similarly, V. Yazıcı1 [10] proposed a coordination framework for scalability and reliability of distributed control plane. But the SMP problem about how to select migrated switches and their target controllers is not solved properly.

B. Heller *et al.* [11] solve how to place the controllers based on propagation latency. But this work only focused on where to place multiple controllers statically. Guang Yao *et al.* [12] consider controller placement problem from the view of the controller load.

To achieve more performance and scalability in large-scale WAN, Md. Faizul Bari *et al.* [13] provide a dynamic controller provisioning framework to adapt the number of controllers and their geographical locations. The framework minimizes flow setup time and communication overhead by solving an integer linear program. But it has to perform a reassignment of the entire control plane based on the collected traffic statistics. This operation easily leads to network instability because it incurs massive state synchronizations. Furthermore, both its greedy and simulated annealing approaches are centralized algorithms which do not adequately utilize the resource of distributed controllers.

To sum up, the existing solutions to dynamic controller provisioning problem (DCPP) are changing the number of controllers and their location via reassigning the switches for controllers. This operation is likely to incur network instability due to a large number of state synchronizations among controllers.

This article has two differences compared with the existing works. First, our pivot is to solve the SMP problem so that we can eliminate the imbalance of the control plane with the available resource. Second, based on the architecture of distributed controllers, we design a synthesizing distributed algorithm that each controller runs its own algorithmic procedure independently.

III. System Model

A. The Motivations

The objective of the switch migration is to serve more network flows under available resource and maximize the resource utility in the control plane. In practice, there are many cases that need to change the mappings. Firstly, since a switch request to a controller may peak at different times, there is an opportunity to increase controller resource utilization by moving more switches into the same controller during off-peak seasons. Then controllers without switches could be shut down or sleep for saving power and communication cost. This can hamper the control plane sprawl, and we call this operation as switches consolidation.

Secondly, once some switches encounters their peak traffic, they can use up all available resources at the controller where

they are placed. One big problem we may encounter is that controllers may become overloaded while other controllers maybe underutilized. in order to improve network performance and resource utilization, a possible configuration is moving some switches from heavy controllers to light ones. We call this operation as load balance.

Thirdly, if all active controllers become overloaded, it is impossible to eliminate hotspots by switch migrations. The operator will deploy some new controllers, and switches will be migrated to such new controllers.

With live switch migration, the switch traffic being served at the same time may be effectively increased by migrating switches with additional resource needed from resource-deficient to resource-rich controllers. However, such migration has to be performed with a well-designed mechanism to fully utilized available resources.

Note that, the preceding three cases could be detected by a load estimation application on the controller. Our algorithm could be used to solve them by designing specific optimal objectives. In the first case, we should design a control power function as payoff function, and then minimize it in such controllers. In the second case, we should design a utility function as payoff function, and then maximize network flows requests under available resource. Essentially, the third one is a special case of second one. For brief, we discuss *load balance* case in the residual part of this paper. Our future work will explore the first case which could toward a green network.

B. SDN Model

Our pivot in this paper is the switch migration problem towards more balanced control plane. So we assume that the SDN controllers have been optimally placed in the distributed topology.

As the literature [12] stated, the load of a SDN controller consists of many factors, such as processing of PACKET_IN events, maintaining the local domain view, communicating with other controllers, as well as installing flow entries. In different scenario, the proportions of those factors differ greatly. But the processing of PACKET_IN events is generally regarded as the most prominent part of the total load [15]. Accordingly, the arriving rate of PACKET_IN events on a controller is counted to measure its load.

Therefore, we give our SDN model as follows. We consider a SDN $\mathcal{G}$ consisting of N controllers $\mathcal{N}=\{c_1,c_2,\cdots,c_N\}$, and M switches $\mathcal{M}=\{s_1,s_2,\cdots,s_M\}$. Let a_i be the control load generated by switch s_i, and d_i be the upper load limits of switch s_i. Accordingly, the switch s_i can be denoted as $s_i:(a_i,d_i)$.

We use $c_j:(A_j,S_j)$ as the controller model. A_j represents the capacity for the controller c_j, and $S_j \subset \mathcal{M}$ denotes the set of switches managed by controller c_j.

IV. Network Utility Maximization Problem in SDN

Our primary objective is to find out how switch migration policies should be employed so that the network utility is maximized. We assume that the more events the controller handles under the available resource, the higher the utilities will be produced. Based on the our SDN model, we formulate switch migration problem as a centralized network utilization maximization problem in SDN,

$$SMP: \quad \max_{S_j} \sum_{c_j \in \mathcal{N}} U_j \tag{1}$$

$$s.t. \quad \sum_{s_i \in S_j} a_i \le A_j, \ \forall c_j \in \mathcal{N} \tag{2}$$

$$a_i \le d_i, \ \forall s_i \in S_j, c_j \in \mathcal{N} \tag{3}$$

$$S_j \bigcap S_{j'} = \varnothing, \forall j \ne j', \ S_j \subseteq \mathcal{M} \text{ and } S_{j'} \subseteq \mathcal{M} \tag{4}$$

$$\bigcup_{c_j \in \mathcal{N}} S_j = \mathcal{M} \tag{5}$$

where U_j is network utility produced by controller c_j.

In SMP, the constraints (2) limit that the total load cannot exceed controller's capacity. The constraints (3) limit the upper bound of each switch to avoid that a small part of switches exhausts controller resource. The constraints (4) ensure that different controllers do not overlap. And the constraints (5) ensure the self-contain of set $\mathcal{M}$.

We believe that once a controller is powered on, the more resource it is possessed, the more utilities it will produce if all the resource consumers are legal. In addition, we assume that network utility function is twice differentiable, increasing and strictly concave. Hence, we define network utility function for $c_j \in \mathcal{N}$ with log function,

$$U_j = \sum_{s_i \in S_j} \log(a_i) \tag{6}$$

Then the objective function can be reformulated as,

$$\max_{S_j} \sum_{c_j \in \mathcal{N}} U_j = \max_{S_j} \sum_{c_j \in \mathcal{N}} \sum_{s_i \in S_j} B_{ij} \tag{7}$$

Where $B_{ij}(t)$ denotes the network utility produced by switch i on the controller j, *i.e.*,

$$B_{ij} = \log(a_i) \tag{8}$$

Under the constraints of SMP, the more load a controller serves, the more network utilities it will produce.

V. Distributed Hopping Algorithms

Theoretically, the problem SMP can be reformulated as 0-1 integer linear program, which is a typically combinatorial network optimization problem, and very difficult to solve. Although we can approach the optimal solution through lagrangian relaxation with quadratic equality constraints and solve its dual problem, or decoupling it into several knapsack problems, it incurs time consuming [16].

Actually, many important network design problem can be formulated as a combinatorial network optimization problem, and a surge of studies have been provided to solve it and have made a significant progress, but many of them are designed to centralized implementations [17] or time-consuming as the network size becomes larger [18][19]. In our scenario, we need a approach that can be concurrently processed in a distributed manner, because each SDN controller manages its local switches and interacts with its neighbors. Moreover, network running the distributed algorithms are more robust to the network dynamics (e.g., switch migration and controller sleep).

In this article, we refer to a markov approximation framework using the log-sum-exp function to approximate the optimal value of our SMP. Based on this, we provide a distributed hopping algorithm in a synthesizing form. In the subsequent section, we first describe the log-sum-exp approximation of SMP. Then we illustrate the detailed design of DHA.

A. Log-Sum-Exp Approximation

Let $\mathcal{H}$ denote the σ-algebra of $\mathcal{M}$, and $\mathcal{H}$ is consisted of all the subsets of $\mathcal{M}$ (including empty set and $\mathcal{M}$), we define a network configuration as a network partition.

Definition 1 Network Configuration $f=\left[S_1^f,\cdots,S_j^f,\cdots,S_N^f\right]$ is a configuration of network $\mathcal{G}$ if

- $S_j^f \in \mathcal{H}$, and $S_j^f \bigcap S_{j'}^f = \varnothing, \forall j \neq j'$,
- $\bigcup_{c_j \in \mathcal{N}} S_j^f = \mathcal{M}$,

where S_j^f denotes the switch set controlled by c_j under the configuration f. Let $\mathcal{F}$ be the set of all feasible f.

Then SMP problem can be rewritten as follows,

$$SMP^f: \max_{f\in\mathcal{F}} \sum_{c_j\in\mathcal{N}} \sum_{s_i\in S_j^f} B_{ij}(f) \tag{9}$$

$$s.t. \sum_{s_i\in S_j^f} a_i(f) \le A_j,\ \forall c_j \in \mathcal{N} \text{ and } f\in\mathcal{F} \tag{10}$$

$$a_i(f) \le d_i,\ \forall s_i \in \mathcal{M} \text{ and } f\in\mathcal{F} \tag{11}$$

where $B_{ij}(f)$ denotes B_{ij} calculated by equation (8) under configuration f. $a_i(f)$ represents control load generated by switch s_i under configuration f. The problem maximizes network utility by choosing a optimal configuration f. However, since the feasible configuration set $\mathcal{F}$ is exponentially large, it is still a NP-hard combinatorial network optimization problem.

Let p_f be the percentage of time that configuration f is in use on configuration space $\mathcal{F}$. An equivalent formulation of the problem SMP^f is as follows,

$$SMP^f - EQ: \max_{p_f\ge 0} \sum_{f\in\mathcal{F}} p_f \sum_{c_j\in\mathcal{N}} \sum_{s_i\in S_j^f} B_{ij}(f) \tag{12}$$

Besides all the constraints of the problem SMP^f, the equation $\sum_{f\in\mathcal{F}} p_f = 1$ is satisfied for the problem $SMP^f - EQ$.

To solve this problem, we use the log-sum-exp function to approximate the optimal value of $SMP^f - EQ$ as follows,

$$\max_{f\in\mathcal{F}} \sum_{c_j\in\mathcal{N}} \sum_{s_i\in S_j^f} B_{ij}(f) \approx \frac{1}{\beta}\log\left(\sum_{f\in\mathcal{F}} \exp\left(\sum_{c_j\in\mathcal{N}} \sum_{s_i\in S_j^f} B_{ij}(f)\right)\right) \tag{13}$$

where β is a positive constant. As the Theorem 1 in literature [14] stated, we solve the approximated version of the problem $SMP^f - EQ$, off by an entropy term $\frac{1}{\beta}\sum_{f\in\mathcal{F}} p_f \log p_f$. So the objective of $SMP^f - EQ$ can be rewritten in approximated form,

$$SMP^f - MA: \max_{p_f\ge 0} \sum_{f\in\mathcal{F}} p_f \sum_{c_j\in\mathcal{N}} \sum_{s_i\in S_j^f} B_{ij}(f) - \frac{1}{\beta}\sum_{f\in\mathcal{F}} p_f \log p_f \tag{14}$$

This additional entropy term opens a new design space for exploration.

Since the objective function of problem $SMP^f - MA$ is twice differentiable, increasing and strictly concave for all p_f, and all the constraints are linear, Karush-Kuhn-Tucker (KKT) conditions are necessary and sufficient for an existing optimal solution. We can conclude that,

Theorem 1 The optimal solution of the problem $UM^f - MA$ is $p_f^*(B)$, $f\in\mathcal{F}$ like that,

- $$p_f^*(B) = \frac{\exp\left(\sum_{c_j\in\mathcal{N}} \sum_{s_i\in S_j^f} B_{ij}(f)\right)}{\sum_{f\in\mathcal{F}} \exp\left(\sum_{c_j\in\mathcal{N}} \sum_{s_i\in S_j^f} B_{ij}(f)\right)}, \forall f\in\mathcal{F}$$

- The optimality gap between SMP^f and $SMP^f - MA$ is bounded by $\frac{1}{\beta}\log|\mathcal{F}|$, where $|\mathcal{F}|$ represents the size of $\mathcal{F}$.

The symbol $B=\left[B_{11}(f),B_{12}(f),\cdots B_{ij}(f)\cdots B_{MN}(f)\right]$.

The proof can refer to our technical report about this paper [21].

B. Distributed Hopping Algorithm Design Based on Markov Chain

As stated in Lemma 1 of the literature [14], there exists at least one continuous-time time-reversible ergodic markov chain with stationary distribution $p_f^*(A)$. The state space $\mathcal{F}$ has to satisfy two conditions. *First*, with the property of ergodicity, any two states in this state space can communicate with each other through at least one path. *Second*, the markov chain must obey detailed balance equation, *i.e.*, $p_f(B)q_{f,f'}(B)=p_{f'}(B)q_{f',f}(B)$ where $q_{f,f'}(B)$ denotes the transition rate from the strategy f to f'.

We define neighbor set $\mathcal{K}(j)$ of c_j as the controllers whose switch directly links to at least one of switch in S_j^f. The transition rate $q_{f,f'}(B)$ will be not zero, if both configurations $f=\left[S_1^f,\cdots,S_{j_1}^f,\cdots,S_{j_2}^f,\cdots,S_N^f\right]$ and $f'=\left[S_1^{f'},\cdots,S_{j_1}^{f'},\cdots,S_{j_2}^{f'},\cdots,S_N^{f'}\right]$ satisfy,

- C1: $\forall c_j\in\mathcal{N}\setminus\{c_{j_1},c_{j_2}\}$, $S_j^f\equiv S_j^{f'}$, $\left|S_{j_1}^f-S_{j_1}^{f'}\right|+\left|S_{j_2}^f-S_{j_2}^{f'}\right|=2$, where $|\bullet|$ represents the element number of set. That means only one switch is migrated from one switch set in f to another in f'.
- C2: $c_{j_2}\in\mathcal{K}(j_1)$, i.e., the switch migration only happens among neighbors.

We define the set $\mathcal{E}(f)$ as $\forall f'\in\mathcal{E}(f)$ satisfies to the above two conditions with f. We can see that the jump from f to f' is scale-limited so that only two correspondingly neighboring controllers change their utilization ratios and take the reminder invariable. Accordingly, let $\mathcal{Q}(f,f')$ denotes the invariable domains under both configuration f and f'. It will appear between both sides of detailed balance equation, we thus ignore it. The transition rate $q_{f,f'}(B)$ in our scenario is formulated as the revised version of OPT 1 in the literature [14], *i.e.*,

$$q_{f,f'}(B)=\left[\left(\exp\left(\sum_{c_j\in\mathcal{N}-\mathcal{Q}(f,f')}\left(\sum_{s_i\in S_j^f}B_{ij}(f)\right)\right)\right)^{-1}\right]_{f'}^{\mathcal{E}(f)} \tag{15}$$

Where the function $[a]_b^c=a$, if $b\in c$ else 0. $q_{f',f}(B)$ can be calculated in a symmetric way. If we limit that the switch only migrated to its neighbors, we can see that the transition rate can be calculated in a local way.

In SDN, there is a logically centralized global view where all controllers share the information. Therefore, each controller can collect fresh value $B_{ij}(f)$ in $\mathcal{N}-\mathcal{Q}(f,f')$ to calculate $q_{f,f'}(B)$ and $q_{f',f}(B)$. Let $q_f(B)=\sum_{f'\in\mathcal{F}}q_{f,f'}(B)$, the network will sojourn in the state f for a period that reduces to the exponential distribution with parameter $q_f(B)$. Based on the theorem 1, we can deduce that,

$$q_f(B)=\left|\mathcal{E}(f)\right|\left(\exp\left(\sum_{c_j\in\mathcal{N}-\mathcal{Q}(f,f')}\left(\sum_{s_i\in S_j^f}B_{ij}(f)\right)\right)\right)^{-1} \tag{16}$$

The configuration f is composed of the local partitions of all individual controllers. The transition from configuration f to f' is taken place by a switch migration from one controller to its neighbor. If each controller counts down a clock and waits for transition until clock terminates, we can design that the transitions are present at two neighbor controllers. Then, the Markov Chain can be calculated in a distributed manner.

We briefly describe the distributed hopping algorithm (DHA) as follows. The following procedure runs on each controller independently, and we focus on a particular controller c_n.

Stage 1: Initially, given an SDN topology with distributed controllers, any controller c_n is allocated a switch domain under the configuration f.

Stage 2: Controller c_n randomly selects a switch s from its domain S_n^f with the size of $|S_n^f|$ and a controller $c_{n'}$ from its neighbor set $\mathcal{K}(n)$. Then c_n will count down a random number. The random number is generated by the exponential distribution with mean $\bar{u}_n^f$ which represents as follows,

$$\bar{u}_n^f=\left(\left|S_n^f\right|\left|\mathcal{K}(n)\right|\right)^{-1}\left(\exp\left(\sum_{c_j\in\mathcal{N}-\mathcal{Q}(f,f')}\left(\sum_{s_i\in S_j^f}\hat{B}_{ij}(f)\right)\right)\right) \tag{17}$$

Stage 3: If the count is expired and no existing switch migration activity in its neighbors is sensed, the controller c_n will broadcast the coming migration activity between c_n and $c_{n'}$ to its neighbors. Then the controller c_n will migrate the selected switch into the controller $c_{n'}$. After migration, the controller c_n will update all utilization ratios $\hat{B}_{ij}(f)$ of the switches in

$\mathcal{N}-\mathcal{Q}(f,f')$ into the network global view, and broadcast it to its neighbors.

Stage 4: Conversely, if there is such an activity between its neighbors, the controller c_n resets the timer. The algorithm returns to Stage 2.

The pseudocode of DHA is shown in Algorithm 1 which runs on each individual controller independently. We focus on a particular controller c_n.

Algorithm Distributed Hopping Algorithm

```
Initialization:
    Initial configuration f , controller c_n with switch set S_n^f ;
    Let c_n 's neighbor set be K(n) ;
    Let k = 0 , tag =0;
End

Procedure Selection( c_n )
    Randomly select a switch s from S_n^f ;
    Randomly select a controller c_n' from K(n) ;
    Acquires total utilization ratios of N-Q(f,f') ;
    Generates a timer with T_n ← rand('exp', u_n^f) ;
    Begin counting down T_n
    while the timer T_n does not expire do
        if sense the existence of switch migration activity then
            tag = 1;
            break;
        end if
    end while
    If tag == 1 then
        Terminates current countdown timer and invoke Section( c_n );
        Tag = 0;
    else
        Controller c_n migrate switch s to controller c_n' ;
        Announces to its neighbors;
    end if
```

We then have the conclusion as follows.

Theorem 2 The process of distributed hopping algorithm is the implementation of time-reversible Markov Chain with stationary distribution $p_f^*(B)$, $\forall f \in \mathcal{F}$.

The proof can refer to our technical report about this paper [21].

VI. IMPLEMENTATION

A. DHA-CON

In this section, we implement a control mechanism prototype atop Beacon controller [26] based on DHA algorithm, called DHA-CON, including a load estimation module, a DHA decision module and a distributed data store module.

Load Estimation. A load estimation module runs as a control application. It tracks the controller loads, and predicts the average message arrive rate from each switch. We set two thresholds, upper limit and lower limit, to indicate whether startup our DHA modules. If the loads are less than lower limit or bigger than upper limit for one minute, load estimation triggers DHA module to switch migration.

DHA Decision. Each controller runs a DHA instance to decide the switch migration. There are two operating models for DHA, the balance one and the green one. First, if a controller is a hot spot, i.e., its loads are bigger than the upper limit for one minute, DHA will work in the balance model to offload part of loads for equilibrium. Second, if a controller is cold spot, i.e., its loads are less than lower limit, DHA will work in the green model to offload all loads and shut down this controller.

Distributed data store. A distributed data store provides a logically central view for controller cluster. It stores all switches information, including data from load estimation module.

B. Controller-to-Controller Interface

We need to extend eastbound and westbound interface so that controllers could communicate with each other during DHA process. As shown in Fig. 1, suppose controller c_j with neighbors $\mathcal{K}(j)$, each runs a DHA thread. When the countdown timer in c_j expires, and there are no existing migration activities in its neighbors, it will emit migrating *request* message to its selected destination c_d. This message includes the migrating switch ID. Then c_d will reply an ACK message to c_j. The controller c_j broadcast *notification* message to its neighbors to suggest that there is a migration activity between c_j and c_d. Finally, the switch migration could be started. We refer the reader to [9] for details of messages needed during switch migration.

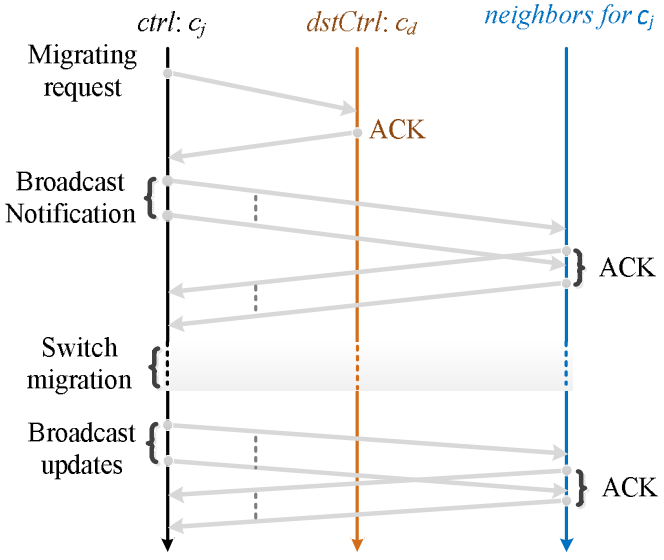


Fig. 1. The message interactions among controllers

After one time of switch migration, controller c_j and c_d update their utilization ratios in the central data store, and broadcast *updates* to their neighbors respectively.

VII. THE NUMERICAL EVALUATION

A. Simulation Setup

In this section, we evaluate the performance of our prototype under the experimental environment shown in Fig. 2. Consider performance interferences between Mininet and controllers, we deploy Mininet [20], Beacon controller and our DHA-CON on different physical machines. Each physical machine runs Ubuntu 12.04 LTS with JDK 1.7.

Instead of the artificial topologies, we use two real network topologies Chinanet [22] (38 nodes and 59 links) and Cernet [23] (36 nodes and 53 links) from zoo topology. Chinanet is a real ISP topology from China Telecom, one of three largest ISPs in China. Cernet is the largest education and research network in China. In addition, we install Beacon controller in an individual machine to simulate single centralized controller. Other five physical machines run DHA-CON instances. All physical machines have exactly the same configuration with 3.4GHz Intel Core i7 processor, 4GB of DDR3 RAM and a 1 Gbps NIC. They are connected by a H3C S5500 switch. We use iperf [24] to generate TCP flows between hosts. To simulate realistic traffic, all flows are generated as traffic characterization described by [25] such as flow size distribution and arriving rate.

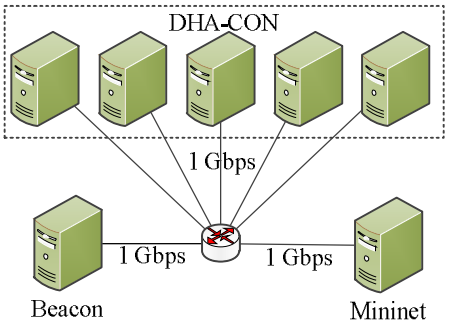


Fig. 2 the experimental topology

We focus on verifying whether our distributed hopping algorithm can improve the performance and scalability of distributed control plane. We compared our DHA-CON (4 controllers) with the scenarios of single controller (S-CNTL), static distributed controllers (D-CNTL) (4 controllers), and Dynamic Controller Provisioning Problem with Greedy Knapsack and Simulated Annealing (DCP-GK and DCP-SA). In DCP-GK or DCP-SA, we initially deploy four controllers, and we will add a new one when one of its controllers overload. Specifically, we first compare average flow setup time along with traffic flows. Then, we evaluate the migration cost caused by DHA. Third, we compare the average utilization ratios of controllers. Finally, we evaluate utility gap of DHA. In our experiment, we define utility gap as the difference between system utility achieved by DHA and the optimal utility obtained by exhaustively searching algorithm which search the feasible configuration space.

B. Parameters Measurement

Before our evaluation, we have to get the values of some parameters in DHA, that is, controller capacity and the upper limits of switch. We use the topology that two physical machines are connected with a switch. One machine runs a Beacon instance, another runs Cbench [27], a program for testing OpenFlow controllers. Each machine has one NIC with 1Gbps.

Cbench works in throughput mode with the command, cbench -c 192.168.1.3 -p 6633 -m 10000 -l 10 -s 16 -M 1000 -t. We find that the average throughput of Beacon is about 1500 kilo requests per second with 4 threads.

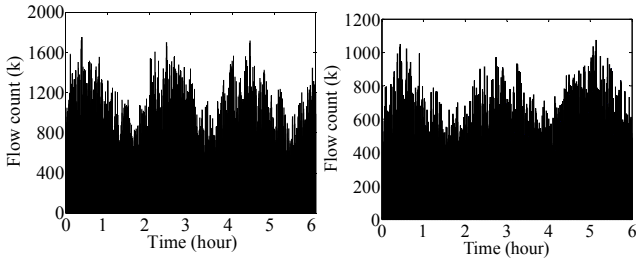
In our experiment, since the switch bandwidth is limited by the loopback interface of Mininet, it is difficult to overload the distributed controllers. So we have to restrict the controller capacity at a low level so that the controller is over-subscribed less than the factors of $\rho = 1:5$ (In datacenter network up-links from ToRs are typically 1:5 to 1:20 oversubscribed [28]). The capacity of each controller in our experiment is limited to 300 kilo requests per second. And the upper limit of switch is simply calculated by equation (18), where k represents the number of switches under the controller, and $\bar{a}_i$ represents average loads generated by switch s_i.

$$d_i = \rho^{-1} \times throughput / k + r(\bar{a}_i) \tag{18}$$

At the right hand of the equation, the first item, called basic item, is calculated based on the controller throughput, control scale and over-subscription ratio. The second item, called individual item, is a random number not beyond $\bar{a}_i$.

C. Numerical Results

Our objective is to increase network utilities, so that they can handle as many OF event requests as possible with their available resources. We set $\beta = 50$ during simulations.



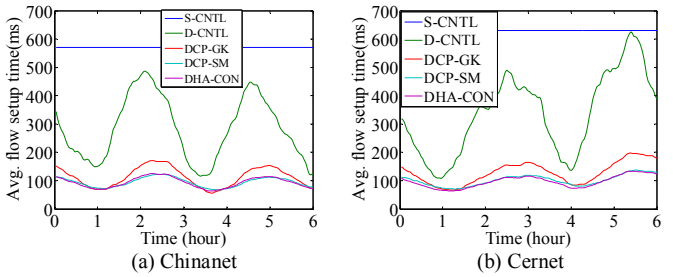
(a) Chinanet (b) Cernet

Fig. 3 Flow count

We run each simulation for 6 hours. Fig. 3 shows the flow counts on Cernet and Chinanet respectively. Simulations are repeated for three times. At each time, we record flow setup time, the number of packets exchanged between controllers, and controller utilizations for different scenarios. We show the average results of three repeated simulations.

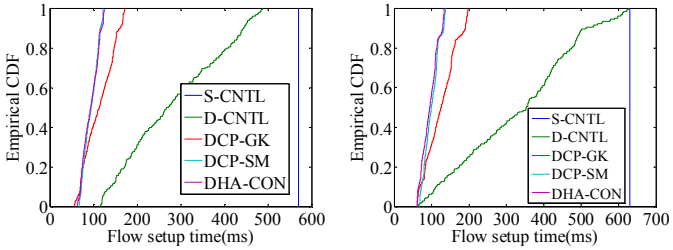
Flow setup time. In the simulation, we use average flow setup time to measure the effect of our DHA-CON, because it reflects controller load changes caused by switch migration.

We compare the average flow setup time of DHA-CON (4 controller instances), S-CNTL, D-CNTL (4 controller instances), DCP-GK and DCP-SM (4~5 controller instances) in two different topologies. Fig. 4 shows their time curves. We can see that S-CNTL is a constant, since single controller is overloaded during the simulation. However, the flow setup time for other four scenarios fluctuates along with flow count, yet has different ranges. That is, D-CNTL has the largest fluctuation, DCP-GK takes second place, and DHA-CON and DCP have less fluctuation.



(a) Chinanet (b) Cernet

Fig. 4 Average flow setup time



(a) Chinanet (b) Cernet

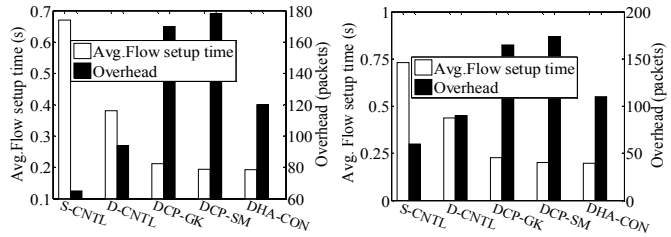
Fig. 5 Empirical CDFs

There are several reasons to explain the above results. First, single controller has the lowest scalability due to its resource-limited architecture. Second, compared with S-CNTL, although D-CNTL has a distributed control plane, its static architecture is likely to induce some heavy controllers that suffer from long flow setup time. Third, DCP-GK and DCP-SM could eliminate overloaded controller via adding a new controller, and DHA-CON achieves such scalability via switch migration. That is, DCP-GK and DCP-SM must run reassignment between controllers and switches while our DHA-CON only migrate several switches from one overloaded controller to light one.

The empirical CDFs in Fig. 5 definitely present that dynamic control plane (i.e., DHA-CON and DCP) are less vulnerable to flow count than D-CNTL.

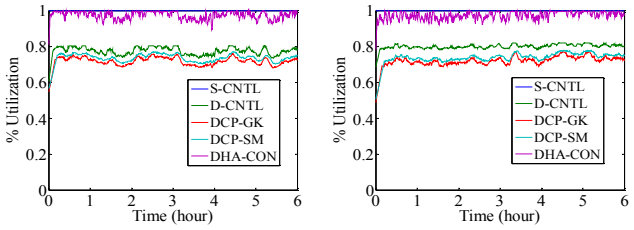
Overhead. We compare the overhead for five scenarios. Fig. 6 presents the communicating overhead and average flow setup time with Chinanet and Cernet topologies. S-CNTL has the lowest communicating overhead because there are no controller-to-controller messages. Compared with S-CNTL, D-CNTL has higher overhead, because controllers need to state synchronization for global network view. DHA-CON generates more packets than D-CNTL, since it needs to migrate switches except state synchronization in DCNTL. But the cost for migration switches is small, no more than 20% of D-CNTL. DCP-GK and DCP-SM have highest communicating overhead. Their cost is twice times that of D-CNTL, and 1.5 times that of DHA-CON, because DCP that reassigns the mappings between controllers and switches will incur more switch migrations than DHA-CON.

As mentioned earlier, average flow setup time for DHA-CON is lowest than other scenarios, about 0.2 s. The average flow setup time for DCP-GK and DCP-SM are close to our DHA-CON, about 0.25 s for Chinanet and Cernet respectively. But the average flow setup time for S-CNTL and D-CNTL are more than 0.65 s and 0.35 s for both topologies.



(a) Chinanet (b) Cernet

Fig. 6 Summary of Overhead and Average Flow Setup Time



(a) Chinanet (b) Cernet

Fig. 7 Average controller utilization

Average utilization ratio. To validate the scalability of DHA-CON, we count the average utilization ratio for different scenarios. As show in Fig. 7, S-CNTL has 100% utilization due to its limited resource. Each controller in DHA-CON has more than 90% utilization. Since DHA-CON balances the loads among controllers so that more requests can be served. Controller in D-CNTL has less than 80% utilization in average because the load imbalance among different controllers. DCP-

GK and DCP-SM have lowest utilization in average. Although DCP could serve the same total loads with DHA-CON, but it needs more controllers than DHA-CON (one more than DHA-CON in our simulation).

Utility gap. Theorem 1 provides a utility loss bound for our DHA algorithm. In the worst case, the Log-sum-exp approximation can lead to utility loss of $\frac{1}{\beta}\log n$, where n is the number of feasible network configurations. In our simulations, we have,

$$\frac{1}{\beta}\log n = \begin{cases} \frac{1}{50}\log 4^{38} = 0.46 \text{ for Chinanet} \\ \frac{1}{50}\log 4^{36} = 0.43 \text{ for Cernet} \end{cases}$$

We obtain the optimal network configuration by exhaustively searching the feasible network configurations. When $\beta = 50$, the average actual utility loss is 0.23 and 0.22 for Chinanet and Cernet. We see that the performance loss bound is guaranteed, and the observed utility loss is quite smaller than the bound.

VIII. CONCLUSIONS

In this paper, we make the first attempt to explore SMP problem for more scalable control plane under the available resource. We first model this problem as a NUM problem from the view of network loads. And then, we design a synthesizing distributed algorithm to solve it. Finally, we implement a prototype and validate it in two real topologies. Of cause, the control load and locality are not the only important factors when choosing target controllers. Resilience is also an important aspect. We will study them in the future.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, *et al.*, "Openflow: enabling innovation in campus networks," SIGCOMM CCR, 2008, pp.1-6.

[2] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. Mckeown, and S. Shenker, "NOX: Towards an Operating System for Networks," in SIGCOMM CCR, 2008.

[3] David Erickson, "The Beacon OpenFlow Controller," In Proc. 1st Workshop on Hot Topics in Software Defined Networking (HotSDN 2013), pages 13-18, Hong Kong, 2013. ACM Press.

[4] A. Tootoonchian and Y. Ganjali, "HyperFlow: A Distributed Control Plane for OpenFlow," in INM/WREN, 2010.

[5] Teemu Koponen, Martin Casado, Natasha Gude, *et al.*, "Onix: a distributed control platform for large-scale production networks," In Proc. OSDI 2010, pages 351-364, Berkeley, 2010. USENIX Association.

[6] Soheil Hassas Yeganeh and Yashar Ganjali. Kandoo: a framework for efficient and scalable offloading of control applications. In Proc. HotSDN 2012, pages 19-24, New York, 2012. ACM Press.

[7] T. Benson, A. Akella, and D. Maltz, "Network traffic characteristics of data centers in the wild," in IMC, 2010.

[8] OpenFlow.https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf

[9] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, R. Kompella, "Towards an Elastic Distributed SDN Controller," In Proc. 1st Workshop on Hot Topics in Software Defined Networking (HotSDN 2013), pages 7-12, Hong Kong, 2013. ACM Press.

[10] V. Yazıcı1, M. Oğuz Sunay1, Ali Ö. Ercan1. Controlling a Software-Defined Network via Distributed Controllers. In NEM submit 2012 arXiv:1401.7651(2012).

[11] B. Heller, Rob Sherwood, and Nick McKeown. The controller placement problem. In Proc. 1st Workshop on Hot Topics in Software Defined Networking (HotSDN 2012), pages 7-12, New York, 2012. ACM Press.

[12] Guang Yao, Jun Bi, Yuliang Li, and Luyi Guo. On the Capacitated Controller Placement Problem in Software Defined Networks. IEEE COMMUNICATIONS LETTERS, 2014.

[13] Md. Faizul Bari, Arup Raton Roy, Shihabur Rahman Chowdhury, Qi Zhang, Mohamed Faten Zhani, Reaz Ahmed, and Raouf Boutaba. Dynamic Controller Provisioning in Software Defined Networks. In CNSM, pp.18-25. 2013.

[14] M. Chen, S. Liew, Z. Shao, and C. Kai, "Markov Approximation for Combinatorial Network Optimization", Proceedings of IEEE INFOCOM 2010, San Diego, CA, US, March, 2010.

[15] A. Tootoonchian, S. Gorbunov, and Y. Ganjali, et al., "On controller performance in software-defined networks," in Proc. of HotICE, 2012.

[16] Y. Feng, B. Li, and B. Li, "Bargaining towards maximized resource utilization in video streaming datacenters," in Proc. of INFOCOM, 2012.

[17] P. Laarhoven and E. Aarts, Simulated annealing: theory and applications. Springer, 1987.

[18] S. Rajagopalan and D. Shah, "Distributed algorithm and reversible network," in Proceedings of CISS, 2008.

[19] J. Liu, Y. Yi, A. Proutiere, M. Chiang, and H. Poor, "Towards Utility optimal Random Access Without Message Passing," Special issue in Wiley Journal of Wireless Communications and Mobile Computing, Dec, 2009.

[20] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In Proceedings of HotNets 2010, pages 19:1–19:6.

[21] DHA-CON-TR-01, http://pan.baidu.com/s/1o67ItiM

[22] http://www.topology-zoo.org/files/Chinanet.gml

[23] http://www.topology-zoo.org/files/Cernet.gml

[24] http://iperf.sourceforge.net.

[25] S. Gebert, R. Pries, D. Schlosser, and K. Heck. "Internet access traffic measurement and analysis", In Traffic Monitoring and Analysis, volume 7189 of LNCS, pages 29–42. 2012.

[26] David Erickson, "The Beacon OpenFlow Controller," In Proc. 1st Workshop on Hot Topics in Software Defined Networking (HotSDN 2013), pages 13-18, Hong Kong, 2013. ACM Press.

[27] ROB SHERWOOD AND KOK-KIONG YAP. Cbench: an OpenFlow Controller Benchmarker. http://www.openflow.org/wk/index.php/Oflops.

[28] Albert Greenberg, James R. Hamilton, Navendu Jain, *et al.*, "VL2: A Scalable and Flexible Data Center Network," in Proc. of SIGCOMM'09, Pages 51-62, Barcelona. Spain. Aug. 2009, ACM Press.