Dynamic SDN Controller Assignment in Data Center Networks: Stable Matching with Transfers

Tao Wang[1] Fangming Liu*[1] Jian Guo[1] Hong Xu[2]
[1]Key Laboratory of Services Computing Technology and System, Ministry of Education,
School of Computer Science and Technology, Huazhong University of Science and Technology
[2]NetX Lab @ City University of Hong Kong

***Abstract*—Software defined networking is becoming increasingly prevalent in data center networks for its programmability that enables centralized network configuration and management. However, since switches are statically assigned to controllers, traffic dynamics cause load imbalance among the controllers. As a result, some controllers are not fully utilized, while switches connected to overloaded controllers may experience long response times. In this paper, we consider dynamic controller assignment so as to minimize the average response time of the control plane. We formulate this problem as a stable matching problem with transfers, and propose a hierarchically two-phase algorithm that integrates key concepts from both matching theory and coalitional games to solve it efficiently. Theoretical analysis proves that our algorithm converges to a near-optimal Nash stable solution within tens of iterations. Extensive simulations show that our approach reduces response time by about 86%, and achieves better load balancing among controllers compared to static assignment.**

I. Introduction

Software defined networking (SDN) has emerged as a new paradigm that shifts network control from distributed protocols to a logically centralized control plane. With its support of flexible network management and rapid deployment of new functionalities, there is an increasing interest in deploying SDN in both inter-data center (e.g., Google B4 [14]) and intra-data center (e.g., Hedera [2]) scenarios.

To improve scalability and avoid a single point of failure [23], the control plane is typically implemented as a distributed system with a cluster of controllers (e.g., Onix [17], DIFANE [27], NVP [16]). Switches are then statically assigned to one or multiple controllers in these systems [17].

However, static assignment between switches and controllers results in long and highly varying controller response times, simply because traffic in data center networks (DCN) fluctuates frequently. Spatially, switches in different layers of the topology experience significantly different flow arrival rates [4], [22]. Temporally, the aggregate traffic usually peaks in daytime and falls at night [12], [22]. Moreover, traffic variability also exists in shorter time scales even when the total traffic remains the same [15]. All these factors cause hot spots among some controllers, leading to excessively long response times for the switches they manage. Although the controller response time may not be significant for elephant flows [2], it fundamentally limits the network's ability to quickly react to changes such as failures and may cause transient congestion to last for a long time [19].

Hence, it is critical to apply dynamic switch assignment to a software defined DCN, for lower controller response time and better utilization of controller resources. Dixit et al. [7] propose an efficient protocol to enable switch migration across multiple controllers without message loss or observable delay. However, the problem of how to determine the assignment remains open. From the switch's perspective, it prefers a controller with low response time to improve performance. From the controller's perspective, it is more willing to manage topologically closer switches to reduce the control traffic overhead. This is important as communication between switches and controllers is frequent and occupies scarce bandwidth resources. These preferences are always intertwined, making the problem especially challenging.

To address these issues, we formulate the dynamic controller assignment (DCA) problem as an optimization problem aiming at minimizing the controller response time and control traffic overhead. In this problem, each controller has a capacity in terms of the request rate it can manage. The switches are dynamically mapped to different controllers when traffic varies. One key challenge is then to develop an efficient solution algorithm to the DCA problem, so that switches can be timely re-assigned in response to variations of network conditions, even in a large-scale DCN.

To this end, we propose a novel two-phase algorithm by casting DCA as a stable matching problem with transfers. In the first phase, we transform the problem into the classical college admissions problem. We define a switch's preference over controllers based on the worst response time that the controller can provide, and a controller's preference based on the control traffic overhead caused by the communication between them. We then obtain a stable matching which guarantees the worst-case response time for switches. This is then used as the initial partition for the coalitional game in the second phase. By breaking from the initial matching and connecting to an underloaded controller, switches participate in the game to achieve a Nash stable solution, where none of them has an incentive to change to a different controller that can lower its

*The Corresponding Author is Fangming Liu (fmliu@hust.edu.cn). The research was supported in part by a grant from National Basic Research Program (973 program) under grant No.2014CB347800, by a grant from The National Natural Science Foundation of China (NSFC) under grant No.61520106005, by a grant from National High Technology Research and Development Program of China (863 program) under grant No.2013AA01A208.

978-1-4673-9953-1/16/$31.00 ©2016 IEEE

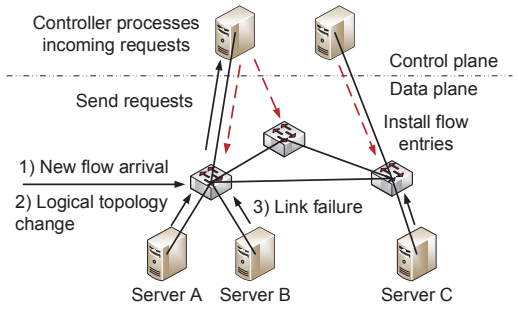

Fig. 1. The DCN model with SDN deployment. The requests sent by switches to controllers come from: 1) new flow arrivals, 2) logical topology changes [16], 3) link failures [19]. Controllers work both reactively and proactively to handle these requests.

response time while not harming others' performance.

The advantages of the two-phase algorithm are two-fold: First, stable matching is competitive of its outcome and efficiency. The deferred acceptance algorithm to generate a stable matching [20] can be easily implemented in a centralized manner with low time complexity, which is suitable for large-scale DCN. Second, the two phases are complementary. The solution of the stable matching phase servers as the input of the coalitional game and accelerates the convergence of the second phase while the coalitional game makes transfers to further improve response time.

Our contributions are as follows:

- We formulate the controller assignment problem as a many-to-one stable matching problem with transfers to reduce both response time and control traffic overhead.
- The proposed two-phase algorithm connects stable matching with utility-based game theoretic solutions. Theoretical analysis proves that the algorithm yields a stable matching in the first phase. In tens of iterations, the second phase can quickly converge to a Nash stable solution that is proved to be within a small constant gap of the optimal solution.
- We carry out trace-driven simulations to show that the two-phase algorithm reduces the convergence steps from above 10000 to 90 compared with directly using coalitional game technique. Our proposed algorithm can reduce response time by 86% and control traffic overhead by 2% on average compared with static assignment. It also achieves near-optimal load balancing among multiple controllers.

II. Model and Formulation

We start by presenting the system model and our problem formulation.

A. Data Center Network Model

Though the physical topology of software defined DCN varies (e.g., Fat-tree, VL2), communication between switches and controllers can be logically viewed as taking place in a two-tier structure between the control and data plane, as shown in Fig. 1.

TABLE I
Key Notations

Symbol	Semantics
s_i	i^{th} switch
c_j	j^{th} controller
α_j	processing capacity of j^{th} controller
β_j	decay factor of j^{th} controller
$\lambda(t)_i$	request rate of i^{th} switch in time slot t
d_{ij}	the hop distance between i^{th} switch and j^{th} controller
$x(t)_{ij}$	whether i^{th} switch is connected to j^{th} controller in time slot t

The control plane consists of M controllers, denoted as $\mathcal{C} = \{c_1, c_2, \cdots, c_M\}$. Their processing capacities are denoted as $\alpha = \{\alpha_1, \alpha_2, \cdots, \alpha_M\}$ in terms of the number of requests it can handle in one time unit. The data plane consists of N switches, $\mathcal{S} = \{s_1, s_2, \cdots, s_N\}$. Virtual machines (VMs) and controllers are hosted on servers. The hop distance between i^{th} switch and j^{th} controller is denoted as d_{ij}. To handle the bursty traffic in DCN, there is a decay factor for each controller (denoted as $\{\beta_1, \beta_2, \cdots, \beta_M\}, \beta_i \in (0,1), i = 1, 2, \cdots, M$) to model the spare capacity. The problem of how to provision enough controllers to satisfy the traffic demand has been studied in [7] and [18]. Here we simply assume that M controllers are sufficient for handling the maximal request rate in a DCN.

The assignment between switches and controllers is denoted as a binary $N \times M$ matrix $\mathcal{X}$. To satisfy the liveness constraint [7], a switch must be exactly connected to one controller as its master. Hence, the sum of the elements in a row of matrix $\mathcal{X}$ is equal to 1. Table I summarizes the key notations for the ease of reference.

B. Controller Response Time Model

Since today's data center topology (e.g., Fat-tree, VL2)can provide high bisection bandwidth, the propagation delay (in μs) in dispatching forwarding rules is less significant than the controller CPU processing time (in ms) [7]. Thus we only model the request processing time on the controller.

We consider a discrete time model where the length of time slot matches the timescale at which switch requests can be precisely recorded. The request demand of the i^{th} switch in slot t is denoted by $\lambda(t)_i$. We assume that the request arrivals follow a Poisson process with $\lambda(t)_i < \alpha_j, \ \forall \ j$.

Switch requests are aggregated at the processing queue of the connected controllers. The load of the j^{th} controller can be represented as:

$$\theta(t)_j = \sum_{i=1}^{N} \lambda(t)_i x(t)_{ij}. \tag{1}$$

We make the following assumptions: (1) $\lambda(t)_i$ are mutually independent. (2) A controller can be modeled as an M/M/1 queue. By applying the Little's law, the average sojourn time

is $\frac{1}{\alpha_j - \theta(t)_j}$. Given that the time of computing single source route is subject to the network size [6], the average processing time of the j^{th} controller can be calculated as below:

$$\vartheta(t)_j = \frac{1}{\alpha_j - \theta(t)_j} O(V^2), \tag{2}$$

where V denotes the number of network nodes (i.e., switches).

The average controller response time in time slot t can be represented as the weighted average of $\{\vartheta(t)_j\}$:

$$\zeta(t) = \frac{\sum_{j=1}^{M} \theta(t)_j \vartheta(t)_j}{\sum_{j=1}^{M} \theta(t)_j}. \tag{3}$$

C. Control Traffic Overhead

Since most SDN deployment uses in-band communications, control traffic competes for the scarce bandwidth [9], [10] with data flows in the network. The default Openflow setting has the switches send asynchronous messages to all controllers that are in the master or equal state. We assume that each request needs to be sent to at least one controller [7]. The control traffic overhead in time slot t can be quantified as:

$$\eta(t) = \sum_{j}^{M} \sum_{i}^{N} d_{ij} x(t)_{ij} \lambda(t)_i. \tag{4}$$

D. Dynamic Controller Assignment Problem

Given a fixed number of controllers, one needs to periodically re-assign switches to controllers to balance the workload according to the dynamic traffic demands.

We now formulate the DCA problem. Our goal is to minimize controller response time while keeping the control traffic overhead low. Hence, we apply a weight factor $\delta \in [0, 1]$ to the response time in the objective function. Mathematically, we have the following formulation:

$$\min \quad \delta\zeta(t) + (1-\delta)\eta(t) \tag{5}$$

$$s.t. \quad \theta(t)_j \leq \beta_j \alpha_j, \ \forall j \tag{6}$$

$$\sum_{j=1}^{M} x(t)_{ij} = 1, \ \forall i \tag{7}$$

$$x(t)_{ij} \in \{0, 1\}, \ \forall i, \ j \tag{8}$$

$$(1), (2), (3), (4).$$

Inequality (6) ensures that no controller is overloaded. The decay factor β_j on the right side forces some capacity to be left aside to handle burst traffic as described before. Constraint (7) ensures that each switch is connected to exactly one master controller at the given time.

This problem is a variant of the multi-objective generalized assignment problem [5] which is NP-hard. Moreover, the problem is large scale in the context of DCN. As the algorithm needs to be computationally efficient for dynamic assignment whenever the network conditions change, traditional methods [3] for solving large-scale multi-objective optimization are computationally prohibitive. In this paper we resort to stable matching and game theory techniques which have been regarded as efficient alternatives for tackling networking problems [25].

III. DCA as a Stable Matching with Transfers

In this section, we show how to transform the DCA problem in two phases: In the first phase, we transform it to a stable matching problem whose solution provides worst-case response time guarantees for each switch. In the second phase, we transform problem to a coalitional game that further reduces the response time. We also discuss about the connections between these two phases, and advantages of using this two-phase model in optimizing the overall performance.

A. The Stable Matching Phase

To formally study the DCA problem, we use the framework of stable matching problem [8] (e.g., college admissions problem) which can be efficiently solved by the deferred acceptance algorithm (DAA) [20]. There are two disjoint sets, namely controllers $\mathcal{C}$ (acting as colleges) that have different capacities to serve requests, and switches $\mathcal{S}$ (acting as students) with distinct request demands. By applying the key concepts of stable matching, namely preferences over the other set and blocking pairs, we deduce the first phase of DCA problem as a many-to-one stable matching problem. Conventionally, we take the notation that $a \succ_c b$ means c prefers a to b.

Switches' objective: Switches seek computation resources to handle the requests. Thus one may build the preference of switch i based on the controller response times defined in Eq. (2). This however introduces some technical difficulty. Particularly, the processing time of controller j in Eq. (2) depends on not only the requests of i^{th} switch, but also those from other switches connected to it.

Most previous work on many-to-one stable matching [8] assumes that the preferences are static and independent of other members. As discussed in [21], finding a stable matching with interdependent preferences is complex, and often requires computing preferences for all of the exponentially many subsets. The sheer complexity of this approach makes it infeasible for our problem. To address this technical challenge, we define a switch's preference over controllers according to the worst response time that the controller can provide.

Clearly, the maximum response delay of the j^{th} controller can be estimated by substituting the maximum load ($\beta_j \cdot \alpha_j$) for $\theta(t)_j$ in Eq. (2):

$$\vartheta(t)_j^{max} = \frac{1}{\alpha_j - \beta_j \cdot \alpha_j}. \tag{9}$$

Definition 1. Switch's preference list over controllers. The preference list of the i^{th} switch s_i is $\Gamma(s_i) = \{c_{j^*}, ..., \}$ which contains controllers whose processing capacity is at least equal to i's request arrival rate. The elements in preference list $\Gamma(s_i)$ are sorted in the ascending order of their worst-case response time according to Eq. (9).

This implies: (1) A controller with larger capacity can serve more switches to better utilize its computing resources. (2) A

switch prefers a controller that can provide lower response time in the worst case.

Controllers' objective: Considering the overhead brought by the switch-to-controller communication, naturally a controller is more willing to accept the switch with smaller control traffic overhead, who will not cause over load of the controller. We thus define the controllers' preferences over switches as below.

Definition 2. Controller's preference list over switches. The preference list of the j^{th} controller c_j is $\Gamma(c_j) = \{s_{i^*}, ..., \}$, with switches whose load does not exceed its capacity, i.e. $\theta(t)_j + \lambda(t)_{i^*} \leq \beta_j \cdot \alpha_j, \forall s_{i^*}$. The elements in the preference $\Gamma(c_j)$ are ranked in an ascending order according to the product of request rate and the hop distance between j^{th} controller and i^* switch, namely $\lambda(t)_{i^*} d_{i^* j}$.

However, different from the traditional college admissions problem [8], our DCA problem has new dimensions that we must take into consideration.

- Every switch has different request rates, while a student only takes up exact one quota.
- Since processing capacity of each controller is not directly related to the number of switches it can serve, the controller's quota depends on which switches it is assigned to.

Intuitively, in a matching Θ, whenever a switch s_i prefers a controller c_j to its currently assigned controller $\Theta(s_i)$, and c_j has the vacant capacity; or by rejecting some lower ranked connected switches, c_j has the vacant capacity, the involved controller and switch then have incentive to break up from its current matching for lower response time and lower control traffic overhead, respectively. Formally, we define the blocking pair as below.

Definition 3. Blocking pair. In a matching Θ, a switch-controller pair (s_i, c_j) is a blocking pair if it satisfies any of the following two conditions:

(1) $c_j \succ_{s_i} \Theta(s_i)$, and $\theta(t)_j + \lambda(t)_i \leq \alpha_j \beta_j$.
(2) $c_j \succ_{s_i} \Theta(s_i)$, and $\theta(t)_j - \sum_{i^*} \lambda(t)_{i^*} + \lambda(t)_i \leq \alpha_j \beta_j$, where $s_i \succ_{c_j} s_{i^*}$ and $\Theta(s_{i^*}) = c_j$.

Depending on whether a blocking pair satisfies condition (1) or (2), we call it is a type-1 or type-2 blocking pair.

Following the convention of [8], we define the stable matching solution:

Definition 4. Stable matching. A matching Θ is said to be stable if there does not exist any blocking pairs.

Having defined the key concepts, we will show how to obtain a stable matching in Sec. IV.

B. Combination of Stable Matching and Coalitional Game

The stable matching framework is efficient and practical to tackle large-scale problems [25], [26]. However, solely applying stable matching may produce an unbalanced matching. In the stable matching shown in the Fig. 2, all switches are connected to controller-2 leaving controller-1 completely idle.

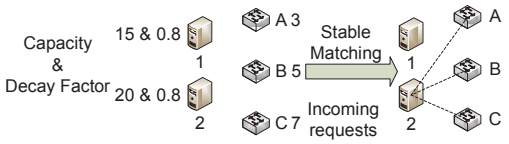

Fig. 2. The distance between switches and controllers is 1 hop. The stable matching solution yields a unbalanced situation that can be further improved by a transfer pair $\langle switch_B \rightarrow controller_1 \rangle$ considering response time.

To address this limitation, we leverage *coalitional game* to improve the solution quality. One problem with coalitional game is that the number of feasible partitions of the switch set $\mathcal{S}$ increases exponentially in N. Thus when traffic demand changes, migrating switches from overloaded controllers needs thousands of iterations as we will show in Sec. V. Fortunately, the outcome of the stable matching process is a mapping Θ defined on the set $\mathcal{S} \bigcup \mathcal{C}$. It satisfies the following conditions: $\forall s \in \mathcal{S}, \forall c \in \mathcal{C}$ 1) $\Theta(s) \in \mathcal{C}$, and 2) $\Theta(c) \in 2^{\mathcal{S}}$. In another word, the stable matching Θ divides $\mathcal{S}$ into M subsets, each of which is associated with one controller. Hence, each subset can be seen as a coalition of switches that are connected to the same controller. In this way, the solution obtained form the stable matching phase can serve as the input of the second phase, and it is suitable for us to use coalitional game theory to further mitigated the unbalanced phenomenon.

C. The Coalitional Game Phase

We now leverage the coalitional game theory to ensure the performance of the mapping between switches and controllers considering response time and traffic overhead.

Since controllers provide computing resources while switches generate requests, the coalitional game among switches is the pair $(\mathcal{S}, \Theta)$, where $\mathcal{S}$ is the set of players (switches) and the coalition $\mathcal{S}_c \subseteq \mathcal{S}$ is the set of switches assigned to the controller $c \in \mathcal{C}$ that can be maintained from Θ. We consider the utility of a switch as the response time from its connected controller as defined in Eq. (2). For lower response time, switches have incentive to negotiate with each other to swap switches based on their utilities.

Note that all switches cannot form a grand coalition due to the limited controller capacity. The switches will form disjoint partitions that are connected to different controllers. This phase can be seen as a coalition formation game [11] in which switches can change their coalitions based on the utility they can get, thus yielding a Nash stable solution in which no switch can improve its utility without harming others'. In order to obtain such a solution, we formally define the transfer rule as below.

Definition 5. Transfer rule. In a matching Θ, a switch s has incentive to transfer from coalition $\mathcal{S}_a$ to $\mathcal{S}_b$ (forming the new coalitions $\mathcal{S}_{a^*} = \mathcal{S}_a \backslash \{s\}$ and $\mathcal{S}_{b^*} = \mathcal{S}_b \bigcup \{s\}$) if it satisfies both of the following:

(1) The transfer does not violate the capacity constraint (6) of controller b^*.
(2) As defined in Eq. (2) and Eq. (4), transfer value satisfies $TV(s, a, b) = \{(1 - \phi) \cdot [\vartheta(t)_{a^*} + \vartheta(t)_{b^*}] + \phi \cdot \eta(t)^{new}\}$-

$\left\{(1-\phi)\cdot[\vartheta(t)_a+\vartheta(t)_b]+\phi\cdot\eta(t)^{old}\right\}<0$. $\phi\in[0,1]$ is a weight factor between response time and overhead.

That is, a transfer is only possible if the controller has enough capacity for handling the requests, and the transfer will enhance the performance of the assignment between switches and controllers considering both response time and traffic overhead as indicated in Eq. (5) has been taken into consideration. Given the matching from the first phase and the transfer rule, the transfer process of the second phase leads to a Nash stable mapping between switches and controllers.

IV. Algorithm Design

So far, we have transformed the DCA problem into a two-phase problem. In this section, we first propose our algorithm based on the famous DAA [20] to solve the first phase and theoretically demonstrate that the algorithm can produce a stable matching. Then based on the transfer rule defined in Definition 5, we devise an algorithm to solve the second phase to improve both response time and overhead. At last, we analyze its optimality and complexity, respectively.

A. Solution for the First Phase

We use the DAA method to generate a stable matching between switches and controllers. We choose switches as the proposing side, which yields a stable matching with the best controller response time for switches among all possible stable matchings.

The algorithm is described in Algorithm 1. Specifically, after each side constructs the preference list based on Definition 1 and 2, switches begin to propose to their most favorable controllers. Each controller receives the proposals, chooses its most preferred switches under the capacity constraint, and reject the rest. The procedure is repeated until no proposal can be made anymore.

Algorithm 1 Stable Matching Phase Procedure

Input: Request rate of each switch during time slot t: $\lambda(t)_i$
Processing capacity of each controller: $\forall j$, α_j
Decay factor of each controller: $\forall j$, β_j
Output: Mapping between switches and controllers: $x(t)_{ij}$, $\forall i,j$

function **StableMatching**(measured request arrival rate $\forall i$, $\lambda(t)_i$, controller capacity $\forall j$, α_j, decay factor $\forall j$, β_j)
  Each switch and controller builds its own preference list as in Eq. (9) and Definition 2: $\forall i$, $\Gamma(s_i)$, $\forall j$, $\Gamma(c_j)$
  repeat
    Each switch proposes to its most preferred controller according to its preference list.
    if All the proposals will not violate the capacity constraint as in Eq. (6) **then**
      The controller temporarily holds all the proposals.
    else
      Based on the controller's preference list, the controller holds the most preferred proposals that will not violate the capacity constraint.
      The controller reject other unacceptable proposals.
    end if
  until No proposals have been made by the switches.
  Transform matching Θ to x_{ij}, $\forall i,j$.
end function

Theorem 1. Algorithm 1 yields a stable matching solution.

Proof: Following the method in [26], we prove the stability of the outcome by contradiction. Since the switch preferences over controllers are only related to processing capacity α_j and decay factor β_j as presented in Eq. (9), the switches' preference lists are the same. In each round, all the switches propose to the same controller, then the controller decides to accept the most preferred switches.

(1) If there exists the type-1 blocking pair (s_{i^*}, c_{j^*}), there must be vacancy in c_{j^*} which contracts the procedure of admitting switches till the processing capacity.
(2) If there exists the type-2 blocking pair, it means that the controller admits less preferred switches in the iterative rounds. However, in each round, the exact one controller receives all the proposals and admits the most preferred ones. Thus the above situation will not happen.

Therefore, the procedure will not cause any blocking pair. The solution obtained by Algorithm 1 is a stable matching as defined in Definition 4. Thus the proof. ■

B. Solution for the Second Phase

Once Algorithm 1 terminates with a stable matching Θ, the corresponding partition $\mathcal{P}=\{\mathcal{S}_1,\mathcal{S}_2,\cdots,\mathcal{S}_M\}$ over $\mathcal{S}$ is used as an initial partition $\mathcal{P}_{initial}$ for the coalitional game in the second phase.

To solve the second phase as a coalitional game, we iteratively find the transfer pair with minimum transfer value which means the lowest social welfare we can get as shown in Algorithm 2.

Algorithm 2 Coalitional Game Phase Procedure

Input: Partition $\mathcal{P}=\{\mathcal{S}_1,\mathcal{S}_2,...,\mathcal{S}_M\}$ obtained from Algorithm 1
Processing capacity of each controller: $\forall j$, α_j
Decay factor of each controller: $\forall j$, β_j
Output: Mapping between switches and controllers: $x(t)_{ij}$, $\forall i,j$

function **CoalitionalFormation**(initial partition $\mathcal{P}$, controller capacity $\forall j$, α_j, decay factor $\forall j$, β_j)
  repeat
    Each switch computes its most preferred transfer.
    Initial transfer pair (s,a,b) with infinity $TV(s,a,b)$ defined in Definition 5.
    for all controllers, $\forall j$, c_j **do**
      Find the transfer pair $(s^*,\Theta(s^*),j)$ with minimum $TV(s^*,\Theta(s^*),j)$.
      if $TV(s,a,b)>TV(s^*,\Theta(s^*),j)$ **then**
        Update $(s,a,b)=(s^*,\Theta(s^*),j)$.
      end if
    end for
    Accept the transfer (s,a,b) and update the partition $\mathcal{P}$.
  until Partition $\mathcal{P}$ converges to a Nash stable partition.
end function

Theorem 2. Given $\mathcal{P}_{initial}$ obtained from Algorithm 1, Algorithm 2 converges to a Nash stable partition $\mathcal{P}_{final}$.

Proof: Denote the partition after k iterations as $\mathcal{P}_k$. Algorithm 2 can be seen as a sequence of partitions:

$$\mathcal{P}_0=\mathcal{P}_{initial}\rightarrow\mathcal{P}_1\rightarrow\mathcal{P}_2\rightarrow\cdots$$

TABLE II
KEY SYMBOLS

Symbol	Semantics
ξ	processing capacity of a controller
$\overline{\theta(t)}$	average request rate on each controller in time slot t
$\varphi(t)_j^{min}$	minimal request rate of switches connected to the j^{th} controller in time slot t
$\mathcal{C}^U$	set of controllers with load larger than $\overline{\theta(t)}$
$\mathcal{C}^L$	set of controllers with load smaller than $\overline{\theta(t)}$

Since a transfer pair (s, a, b) only influences the partition set $\mathcal{S}_a$ and $\mathcal{S}_b$ (forming the new coalitions $\mathcal{S}_{a^*} = \mathcal{S}_a \backslash \{s\}$ and $\mathcal{S}_{b^*} = \mathcal{S}_b \bigcup \{s\}$), every adjacent transfer, e.g., $\mathcal{P}_l \rightarrow \mathcal{P}_{l+1}$, forms an order:

$$\begin{aligned}&(1-\phi)\cdot[\vartheta(t)_{a^*}+\vartheta(t)_{b^*}]+\phi\cdot\eta(t)^{new}\\&<(1-\phi)\cdot[\vartheta(t)_a+\vartheta(t)_b]+\phi\cdot\eta(t)^{old}\end{aligned}$$

which is transitive and irreflexive. As the number of partitions of set $\mathcal{S}$ is finite, Algorithm 2 converges to a final partition $\mathcal{P}_{final}$.

Moreover, suppose that $\mathcal{P}_{final}$ is not Nash stable, and there exists a transfer pair (s, a, b) that can result in better social welfare. Algorithm 2 will then continue, and $\mathcal{P}_{final}$ does not converge. However, this contradicts the previous demonstration of Algorithm 2's convergence.

Therefore, Algorithm 2 converges to a Nash stable partition. ■

C. Optimality Analysis

We now theoretically prove SMT's performance on response time compared with the optimal solution. We consider a simple situation where the capacity of controllers is identical. Key symbols are listed in Table II.

Since Eq. (3) is convex for each controller, according to Jensen's inequality, it is easily can be proved that the minimal response time is achieved when the incoming requests are equally distributed among all the controllers:

$$\zeta(t)^{min} = \frac{O(V^2)\cdot M\cdot\frac{\overline{\theta(t)}}{\xi-\overline{\theta(t)}}}{\sum_{i=1}^{M}\theta(t)_i}. \tag{10}$$

For simplicity, let $\varphi(t)_j^{min}$ denote the minimal request rate of switches connected to j^{th} controller. Hence, when SMT terminates, the difference of load between any two controllers is less than the load caused by switch with the minimal request rate on the higher-loaded controller, mathematically,

$$\theta(t)_j - \theta(t)_i < \varphi(t)_j^{min}, \forall\ i,\ j.$$

For such controllers that $\theta(t)_j > \overline{\theta(t)} > \theta(t)_i$, we have $\theta(t)_j < \theta(t)_i + \varphi(t)_j^{min} < \overline{\theta(t)} + \varphi(t)_j^{min}$. Hence, $\theta(t)_j - \overline{\theta(t)} < \varphi(t)_j^{min}$ is established.

When SMT converges to a Nash stable solution, we divide the $\mathcal{C}$ into two mutually complementary subsets, $\mathcal{C}^U$ in which $\theta(t)_i \geq \overline{\theta(t)},\ \forall\ i \in \mathcal{C}^U$ and $\mathcal{C}^L$ in which $\theta(t)_{i^*} < \overline{\theta(t)},\ \forall\ i^* \in \mathcal{C}^L$, respectively.

We now derive the gap between SMT's solution and the optimal one in Eq. (10) as follows:

$$\begin{aligned}\Delta(\zeta(t)) &= \frac{O(V^2)}{\sum_{i=1}^{M}\theta(t)_i}\left\{\sum_{i=1}^{M}\left[\frac{\theta(t)_i}{\xi-\theta(t)_i}-\frac{\overline{\theta(t)}}{\xi-\overline{\theta(t)}}\right]\right\}\\&< \frac{O(V^2)}{\sum_{i=1}^{M}\theta(t)_i}\sum_{i\in\mathcal{C}^U}\frac{\xi(\theta(t)_i-\overline{\theta(t)})}{(\xi-\theta(t)_i)(\xi-\overline{\theta(t)})}.\end{aligned}$$

Denote $\theta(t)_i - \overline{\theta(t)}$ as $\tau(t)_i$ which satisfies $\tau(t)_i < \varphi(t)_i^{min}$.

$$\Delta(\zeta(t)) < \frac{O(V^2)}{\sum_{i=1}^{M}\theta(t)_i}\sum_{i\in\mathcal{C}^U}\frac{\xi\tau(t)_i}{(\xi-\overline{\theta(t)}-\tau(t)_i)(\xi-\overline{\theta(t)})}.$$

Note that the function on the right side of the above inequality is monotone increasing as $\tau(t)_i$ increases. Hence,

$$\begin{aligned}\Delta(\zeta(t)) &< \frac{O(V^2)}{\sum_{i=1}^{M}\theta(t)_i}\sum_{i\in\mathcal{C}^U}\frac{\xi\varphi(t)_i^{min}}{(\xi-\overline{\theta(t)}-\varphi(t)_i^{min})(\xi-\overline{\theta(t)})}\\&< \frac{O(V^2)}{\sum_{i=1}^{M}\theta(t)_i}\cdot\frac{M\cdot\xi\cdot\varphi(t)^{max}}{(\xi-\overline{\theta(t)}-\varphi(t)^{max})(\xi-\overline{\theta(t)})}\\&\qquad,\ \varphi(t)^{max}=\max(\varphi(t)_i^{min}, i\in\mathcal{C}^U).\end{aligned}$$

Thus, the performance on response time of SMT is within a factor $(1+\frac{\Delta(\zeta(t))}{\zeta(t)^{min}} = 1+\frac{\xi\cdot\varphi(t)^{max}}{\overline{\theta(t)}\cdot(\xi-\overline{\theta(t)}-\varphi(t)^{max})})$ of the optimal value.

Remark: The bound above is relevant to the request demands on the controllers in the time slot t where ξ and $\overline{\theta(t)}$ are known. Since the request rates of switches are far smaller than the processing capacity of controllers, $\varphi(t)^{max}$ in Eq. (11) can be regarded as a small number, which means that the bound is a small constant in each time slot.

D. Complexity Analysis

We now give a brief complexity analysis for both algorithms presented above. Note that the statistical states collected from the data plane can be stored in the control plane. Thus, both algorithms can be executed in the control plane with little synchronization overhead.

For Algorithm 1, as we have discussed in Sec. IV, all the switches have the same preferences over controllers. Thus, Algorithm 1 terminates in M iterations. In each iteration, one controller accepts its most preferred switches. Sorting the switches need $O(Nlog_2(N))$ computation. Therefore the time complexity of Algorithm 1 is $O(MNlog_2(N))$.

For Algorithm 2, the complexity comes from searching for necessary transfers where, from the switches' perspective, the search space is $O(NM)$ in each round.

Considering that M, the number of controllers, is far smaller than the number of switches N and the upper-bound proven in previous subsection, our proposed two-phase algorithm can find a near-optimal solution in efficient time which grows nearly linearly with the size of network increasing as we will shown in Sec. V.

V. Evaluation and Analysis

In this section, we conduct trace-driven simulations to evaluate the performance of our proposed algorithms, hereafter referred to as the Stable Matching with Transfers (SMT) algorithm.

A. Simulation Setup

Topology: We conduct our simulations using the widely adopted fat-tree and VL2 topologies as shown in Fig. 3 and Fig. 4, respectively. For fat-tree, the number of pods is 24 with a total of $3,456$ hosts and 720 switches. For VL2, the degree of intermediate switch (D_I) and the degree of aggregate switch (D_A) are set to 48 with 576 racks each hosting 10 hosts, and $N = 48 \cdot (48+6)/4 = 648$. These numbers are comparable to the size of a commercial data center [4]. Hence, we set the $O(V^2)$ as 10^4 in Eq. (2). Also, we set the number of controllers M to 18, and the capacity of each controller is $1800K$ flows/s [7], which is enough to handle the maximum request rate in our simulations.

Trace: The request arrival rate follows the flow inter-arrival time distribution measured in a real-world data center [4] since a request to the controller is directly triggered by a flow going through a switch. We use κ as a load factor to evaluate SMT in different load conditions.

Schemes Compared: (1) SMT: Our algorithm with two phases. In SMT, unless stated otherwise, ϕ in Definition 5 is set to 0 as we regard response time as the sole objective. The decay factor β_j is randomly chosen between 0.9 and 1. **(2) DCP-GK**: State-of-the-art algorithm from [3] that combines Greedy Knapsack with Simulated Annealing heuristic. We just use the greedy algorithm of DCP-GK since the full algorithm is designed for controller provisioning in WANs where time complexity is less of an issue. **(3) CG**: Directly solving the DCA problem as a coalitional game using a purely combinatoric algorithm which follows the second phase of our SMT but the initial mapping is generated randomly. We will discuss the connection between our SMT and this algorithm. **(4) SM**: Static matching between switches and controllers used as the baseline. The static matching is obtained by DCP-GK after the first run.

We derive the distance matrices for the fat-tree and VL2 topologies, respectively. For fat-tree:

$$d_{ij}^{\text{Fat-tree}} = \begin{cases} 1, & \text{Edge } s_i, \text{ directly connected with } c_j \\ 2, & \text{Agg } s_i, \text{ same pod with } c_j \\ 3, & \text{Edge } s_i, \text{ same pod, not directly connected} \\ 3, & \text{Core } s_i \\ 4, & \text{Agg } s_i, \text{ not same pod with } c_j \\ 5, & \text{Edge } s_i, \text{ not same pod with } c_j \end{cases}$$

For VL2, the traffic which leaves ToR switches always goes through the intermediate switches due to valiant load balancing. Thus:

$$d_{ij}^{\text{VL2}} = \begin{cases} 1, & \text{ToR } s_i, \text{ directly connected with } c_j \\ 2, & \text{Agg } s_i, \text{ same ToR connected with } c_j \\ 3, & \text{Intermediate } s_i \\ 4, & \text{Agg } s_i, \text{ not same ToR connected with } c_j \\ 5, & \text{ToR } s_i, \text{ not directly connected with } c_j \end{cases}$$

To see whether SMT can balance the load among controllers and reduce controller response time, we run all algorithms in discrete time slots during which the request arrival rate changes according to its distribution over time.

B. Effectiveness of *SMT*

Response Time. Fig. 5 and 6 plot the comparison of SMT, DCP-GK and SM in terms of average response time in fat-tree and VL2. We make the following observations: (1) As the total request rate increases, response time also increases since the computing resource on controllers is limited. (2) Request dynamics may cause a sudden increase of response time for SM as shown in Fig. 5. In the extreme case, the response time of SM is 10x that of SMT. It demonstrates that static controller assignment results in severe load imbalance. In our simulations such severe phenomenon happens about 4% of the time in all runs. SMT reduces response time by 86% on average compared with SM. (3) SMT outperforms DCP-GK and reduces the response time by 22% on average. (4) Compared with the optimal response time, SMT is only 0.35% slower on average.

Control Traffic Overhead. Fig. 7 and Fig. 8 depict the control traffic overhead of the three algorithms, where all the overheads are normalized to SM's performance (approximately $7.4e7$). In Fig. 7, it clearly shows that SMT performs the best. Although we assign zero weight to overhead in SMT, the procedure of stable matching phase takes overhead into consideration when controllers evaluate proposals made by switches. Thus, SMT has about 2% less overhead compared with SM on average. However, in Fig. 8, SMT does not always outperform SM in VL2 since control traffic that leaves ToR switches always goes through the intermediate switches due to valiant load balancing, making switches communicate in a longer way with light-loaded controllers.

Load Distribution of Controllers. To show the load distribution among multiple controllers along time, we plot the Jain's fairness index [13] (a value ranges from $\frac{1}{M}$ to 1) of these algorithms in Fig. 9 and Fig. 10. It can be seen that SMT achieves a more balanced distribution under dynamic traffic loads. While the other three algorithms are highly sensitive to traffic variations, the fairness index of SMT remains close to 1, which implies near-optimal load balancing. Fig. 9 and Fig. 10 also plot the max/average/min loads on controllers for SMT. It clearly shows that the load differential among multiple controllers is small.

Traffic Condition and Scalability. Fig. 11 and Fig. 12 show the performance of SMT in terms of response time under

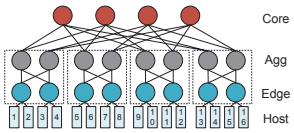

Fig. 3. 4-pod fat-tree (4 core, 8 aggregate and 8 edge switches, respectively) with controllers deployed on the hosts.

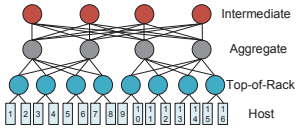

Fig. 4. VL2 (D_I=4, D_A=8, 4 intermediate, 4 aggregate, 8 ToR switches, respectively) with controllers deployed on the hosts.

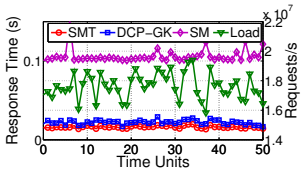

Fig. 5. In fat-tree, SM experiences about 7x controller response time compared with SMT.

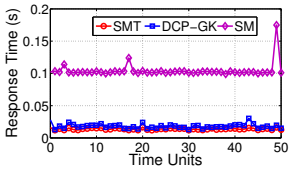

Fig. 6. In VL2, SM experiences about 8x controller response time compared with SMT.

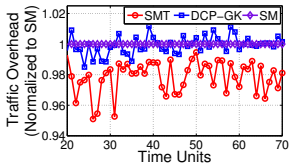

Fig. 7. Normalized traffic overhead in fat-tree.

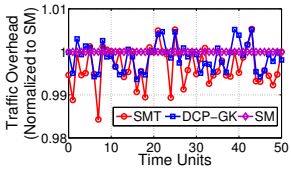

Fig. 8. Normalized traffic overhead in VL2.

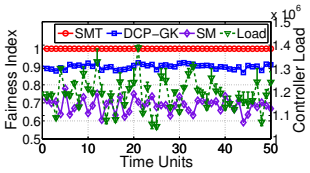

Fig. 9. Controller fairness index in fat-tree.

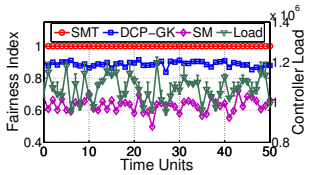

Fig. 10. Controller fairness index in VL2.

different traffic loads. It shows that response time of SMT increases slightly when loads increase, while SM increases sharply since the overload phenomenon becomes severer. In a related thread, to show SMT's scalability, we plot the runtime of SMT under different topologies in Fig. 13. We set the number of pods in fat-tree to 24, 16, 8 with 720, 320, 80 switches, respectively, and set the degree of aggregate switches in VL2 to 48, 32, 24, 16 with 648, 304, 180, 88 switches, respectively. It shows that the runtime of SMT increases linearly with the size of the network and only takes 0.2s to finish in a network with size comparable to a commercial datacenter.

C. Insights of *SMT*

Are transfers necessary? One may ask why we need the second coalitional game phase and whether the transfers are necessary. Fig. 14 and Fig. 15 plot the difference between SMT and stable matching with no transfers (denoted as SMNT) in terms of response time and overhead. For overhead, the transfers with transfer rule defined in Definition 5 bring about 4% more overhead to the network. However, they reduce the response time by about 62% compared with SMNT. This validates transfers' effectiveness to further reduce response time while incurring little more overhead to the network.

The impact of the stable matching phase. To quantify the impact of the stable matching phase on the performance of SMT, we compare SMT with the one-phase algorithm, namely, coalitional game (denoted as CG). Fig. 16 plots the response time normalized to CG's performance. Since the coalitional game phase in SMT begins with the matching obtained from the stable matching phase, its search space is limited, which hinders SMT's ability to further reduce response time as we have shown in Fig. 9 and Fig. 10. In Fig. 16, we do find that the SMT experiences longer response time compared with CG, however the maximum difference is only 0.24 ms. In terms of control traffic overhead, Fig. 17 shows that SMT incurs up to 20% lower overhead compared with CG, since the stable matching phase takes overhead into account. Moreover, the stable matching phase speeds up the convergence of the coalitional game phase as shown in Fig. 18. Specifically, it shows that SMT is able to converge within 90 iterations for 80% of the time for over 300 runs, and the fastest run uses only 44 iterations to converge. CG, on the other hand, takes over 10000 iterations to converge on average.

VI. Related Work

In this section, we survey the state of the art of SDN controller assignment, and the use of stable matching and coalitional game in computer networking.

Dynamic Controller Assignment. To improve robustness and scalability, several works [17], [27] develop a distributed control plane across a cluster of controllers. Nevertheless, the static mapping between switches and controllers may cause hot spots, which motivates dynamic controller assignment. Based on OpenFlow v1.3 [1], Dixit et al. [7] firstly propose a live switch migration protocol which can ensure liveness and safety of DCA while introducing little overhead to the network. Bari et al. [3] present an algorithm to dynamically and efficiently provision controllers in a WAN by periodically reassigning switches to controllers. However, the proposed heuristic is time-consuming and can hardly be applied to the highly bursty DCN. Krishnamurthy et al. [18] propose an elastic controller assignment mechanism by partitioning application states and exploring the dependency between switches and applications. It does not consider the processing time on controllers which is a major cost in flow setup time [7]. By leveraging the migration protocol [7], our work aims at mitigating the load imbalance among controllers and reducing both response time and control traffic overhead.

Stable Matching and Coalitional Game. Stable matching is first introduced by Gale et al. [8] in the marriage problem and has been widely used in the National Resident Matching Program for medical school students for several decades. In the field of networking, Xu et al. [25] advocate for applying stable matching as a general methodology, just like optimization, to tackle networking problems. They apply the framework for the VM allocation problem in [24] to improve performance of VMs and reduce load imbalance on servers. They further extends the classical matching theory for general resource

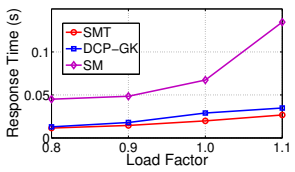

Fig. 11. Response time under different traffic loads in fat-tree.

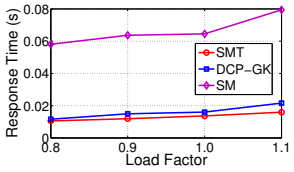

Fig. 12. Response time under different traffic loads in VL2.

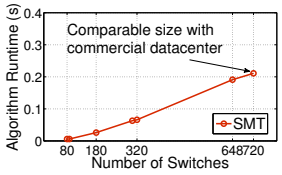

Fig. 13. Runtime of SMT under different topology scales.

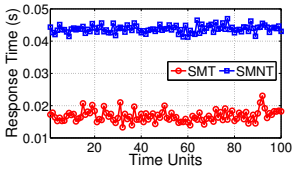

Fig. 14. Response time comparison between SMT and SMNT in fat-tree.

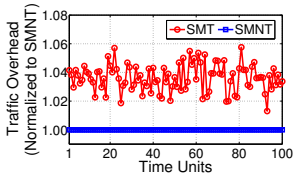

Fig. 15. Traffic overhead comparison between SMT and SMNT in fat-tree.

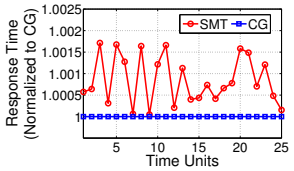

Fig. 16. Response time comparison between SMT and CG in fat-tree.

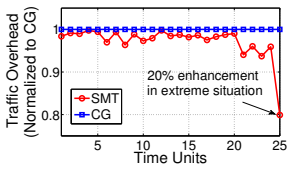

Fig. 17. Traffic overhead comparison between SMT and CG in fat-tree.

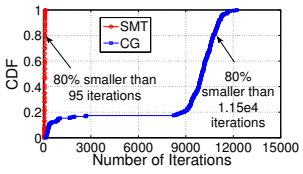

Fig. 18. The CDF of number of iterations in SMT and CG, respectively.

management in cloud computing with VM size heterogeneity in [26].

Game theory is one of the most widely used techniques in networking community. Han et al. [11] present a survey for the use of game theory in communication networks. Specifically, the assignment of users to access points in wireless networks is formulated and solved as a coalitional game. In our work, we leverage these two theories to formulate the DCA problem as a stable matching problem with transfers.

VII. Conclusion

In this paper, we studied the dynamic controller assignment (DCA) problem as a stable matching problem with transfers to minimize the controller response time. The DCA problem is solved in a two-phase manner. First a stable matching is efficiently generated between switches and controllers, which guarantees the response time in worst case. It serves as an input to the second coalitional game phase to further reduce the response time. Theoretical analysis proves that the proposed two-phase algorithm converges to a Nash stable solution. Trace-driven simulation shows that the stable matching phase accelerates the convergence of the coalitional game phase, by reducing the average number of iterations from above 10000 to around 90. The two-phase algorithm, which achieves near-optimal load balancing among controllers, reduces the controller response time by about 22% and 86% compared with state of the art DCP-GK and the simple static assignment, respectively.

References

[1] Openflow specification v1.3.

[2] M. Al-Fares, S. Radhakrishnan, et al. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *Proc. USENIX NSDI*, 2010.

[3] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba. Dynamic Controller Provisioning in Software Defined Networks. In *Proc. IEEE CNSM*, 2013.

[4] T. Benson, A. Akella, and D. A. Maltz. Network Traffic Characteristics of Data Centers in the Wild. In *Proc. ACM IMC*, 2010.

[5] R. Cohen, L. Katzir, and D. Raz. An Efficient Approximation for the Generalized Assignment Problem. *Information Processing Letters*, 2006.

[6] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1959.

[7] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella. Towards an elastic distributed SDN controller. In *Proc. ACM HotSDN*, 2013.

[8] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 1962.

[9] J. Guo, F. Liu, X. Huang, J. Lui, et al. On efficient bandwidth allocation for traffic variability in datacenters. In *Proc. IEEE INFOCOM*, 2014.

[10] J. Guo, F. Liu, D. Zeng, et al. A cooperative game based allocation for sharing data center networks. In *Proc. IEEE INFOCOM*, 2013.

[11] Z. Han. *Game Theory in Wireless and Communication Networks: Theory, Models, and Applications*. Cambridge University Press, 2012.

[12] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. ElasticTree: Saving Energy in Data Center Networks. In *Proc. USENIX NSDI*, 2010.

[13] R. K. Jain, D.-M. W. Chiù, and W. R. Hawe. A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems. 1984.

[14] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al. B4: Experience with a Globally-Deployed Software Defined WAN. In *Proc. ACM SIGCOMM*, 2013.

[15] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The Nature of Data Center Traffic: Measurements & Analysis. In *Proc. ACM IMC*, 2009.

[16] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, N. Gude, P. Ingram, et al. Network Virtualization in Multi-tenant Datacenters. In *Proc. USENIX NSDI*, 2014.

[17] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, et al. Onix: A Distributed Control Platform for Large-scale Production Networks. In *Proc. USENIX OSDI*, 2010.

[18] A. Krishnamurthy, S. P. Chandrabose, and A. Gember-Jacobson. Pratyaastha: An Efficient Elastic Distributed SDN Control Plane. In *Proc. ACM HotSDN*, 2014.

[19] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter. Traffic Engineering with Forward Fault Correction. In *Proc. ACM SIGCOMM*, 2014.

[20] A. E. Roth. Deferred Acceptance Algorithms: History, Theory, Practice, and Open Questions. *International Journal of Game Theory*, 2008.

[21] A. E. Roth and M. A. O. Sotomayor. *Two-sided Matching: A Study in Game-theoretic Modeling and Analysis*. Cambridge University Press, 1992.

[22] A. Roy, H. Zeng, J. Bagga, et al. Inside the Social Network's (Datacenter) Network. In *Proc. ACM SIGCOMM*, 2015.

[23] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood. On Controller Performance in Software-Defined Networks. In *Proc. USENIX HotICE*, 2012.

[24] H. Xu and B. Li. Egalitarian Stable Matching for VM Migration in Cloud Computing. In *Proc. IEEE INFOCOM Workshop on Cloud Computing*, 2011.

[25] H. Xu and B. Li. Seen as Stable Marriages. In *Proc. IEEE INFOCOM*, 2011.

[26] H. Xu and B. Li. Anchor: A Versatile and Efficient Framework for Resource Management in the Cloud. *IEEE Transaction on Parallel Distributed System*, 2013.

[27] M. Yu, J. Rexford, M. J. Freedman, and J. Wang. Scalable flow-based networking with DIFANE. In *Proc. ACM SIGCOMM*, 2010.