An Efficient Online Algorithm for Dynamic SDN Controller Assignment in Data Center Networks

Tao Wang, Fangming Liu, *Senior Member, IEEE*, and Hong Xu, *Member, IEEE, ACM*

***Abstract*—Software defined networking is increasingly prevalent in data center networks for it enables centralized network configuration and management. However, since switches are statically assigned to controllers and controllers are statically provisioned, traffic dynamics may cause long response time and incur high maintenance cost. To address these issues, we formulate the dynamic controller assignment problem (DCAP) as an online optimization to minimize the total cost caused by response time and maintenance on the cluster of controllers. By applying the randomized fixed horizon control framework, we decompose DCAP into a series of stable matching problems with transfers, guaranteeing a small loss in competitive ratio. Since the matching problem is NP-hard, we propose a hierarchical 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 online approach reduces total cost by about 46%, and achieves better load balancing among controllers compared with static assignment.**

***Index Terms*—Software defined networking, data center network, online algorithm, stable matching, coalitional game.**

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 [42], there is an increasing interest in deploying SDN in both inter-data center (e.g., Google B4 [25]) and intra-data center (e.g., Hedera [6]) scenarios. To improve scalability and avoid a single point of failure [39], the SDN control plane is typically implemented as a distributed system with a cluster of controllers (e.g., Onix [28], DIFANE [48], NVP [27]). Switches are then statically assigned to one or multiple controllers [28].

However, static assignment between switches and controllers results in long and highly varying controller response times, since traffic in data center networks (DCN) fluctuates frequently. Spatially, switches in different layers of the topology experience significantly different flow arrival rates [10], [38] and traffic variability [20], [21], [34]. Temporally, the aggregate traffic usually peaks in daytime and falls at night [23], [38]. Moreover, traffic variability also exists in shorter time scales even when the total traffic remains the same [26]. 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 [6], it fundamentally limits the network's ability to quickly react to events such as failures and may cause transient congestion to last for a long time [18], [35].

Further, maintaining a cluster of controllers needs special care considering their synchronization cost, since this synchronization among controllers directly affects the scalability and management of the control plane [32]. Consistent network states should be maintained, otherwise the network performance may significantly degrade [32]. The synchronization among controllers requires all-to-all communications [9], [32], which means the more controllers there are, the more the maintenance costs.

Hence, it is critical to apply dynamic controller provisioning and assignment to a software defined DCN, for lower controller response time and better utilization of controller resources. Dynamic switch migration across controllers is technically feasible as demonstrated by past work such as [16].

We formulate the dynamic controller assignment problem (DCAP) as an online optimization problem aiming at minimizing the total cost. In this problem, each controller has capacity in terms of the maximum request rate it can manage. The switches are dynamically mapped to controllers when traffic varies. One key challenge is then to develop an efficient solution algorithm, so that switches can be re-assigned in a timely fashion in response to variations of network conditions, even in a large-scale DCN.

To solve the long-term DCAP online, we apply the Randomized Fixed Horizon Control (RFHC) framework to decompose the long-term optimization into a series of one-time-slot assignment problems. However, even in each time slot, the assignment problem remains challenging. From the switch's perspective, it prefers a controller with lower 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.

Manuscript received September 10, 2016; revised January 30, 2017 and March 29, 2017; accepted May 6, 2017; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Mascolo. This work was supported in part by the National 973 Basic Research Program under Grant 2014CB347800, in part by NSFC under Grant 61520106005, and in part by the Hong Kong RGC under Grant GRF-11202315 and Grant CRF-C7036-15G. *(Corresponding author: Fangming Liu.)*

T. Wang and F. Liu are with the Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: wta0@hust.edu.cn; fmliu@hust.edu.cn).

H. Xu is with the Department of Computer Science, City University of Hong Kong, Hong Kong (e-mail: henry.xu@cityu.edu.hk).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors. This consists of a PDF (0.091 MB) appendix containing proofs to Theorems 1–6 in the paper.

Digital Object Identifier 10.1109/TNET.2017.2711641

These preferences are always intertwined, making the problem especially challenging.

To this end, we propose a novel two-phase algorithm by casting the assignment problem as a stable matching problem with transfers. In the first phase, we transform the problem into the classical college admissions problem [17]. By defining 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 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 response time while not harming others' performance.

The advantages of the two-phase algorithm are two-fold: First, stable matching is competitive in its outcome and efficiency. The deferred acceptance algorithm to generate a stable matching [36] 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 serves 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.

Specifically, our contributions are as follows:

- We formulate the DCAP as an online cost minimization over a long time frame. By leveraging the RFHC framework, we decompose the DCAP into a series of one-time-slot assignment problems. In each time slot, our proposed algorithm, which connects stable matching with utility-based game theoretic solutions, can quickly converge to a Nash stable solution within a small constant gap of the optimal solution. The overall competitive ratio of our online framework is $\frac{f(1+\frac{\xi\cdot\nu}{\kappa\cdot(\xi-\kappa-\nu)})}{f(1)}(1+\frac{\delta}{(1+\omega)\cdot\psi})$, where $f(\cdot)$, δ and ψ are system-dependent parameters.
- We carry out trace-driven simulations to show that the two-phase algorithm reduces the number of iterations to achieve convergence from above 10000 to 90 compared to directly using coalitional game techniques. In the single time slot setting, our two-phase algorithm can reduce response time by 65% on average compared with static assignment. It also achieves near-optimal load balancing among controllers. In the online setting, our online algorithm can reduce the total cost by 46.2% compared with static assignment.

The rest of the paper is organized as follows. In Sec. II, differing from our preliminary work [41], we reformulate the DCAP as an online optimization to minimize the total cost. In Sec. III, by applying RFHC, we present a new online framework together with the two-phase algorithm in [41] to solve DCAP. Further, we prove the competitive ratio of the proposed algorithm, along with an analysis of its time complexity. In Sec. IV, we conduct trace-driven simulations to evaluate the performance of the proposed online framework. Sec. V discusses related work, and finally Sec. VI concludes.

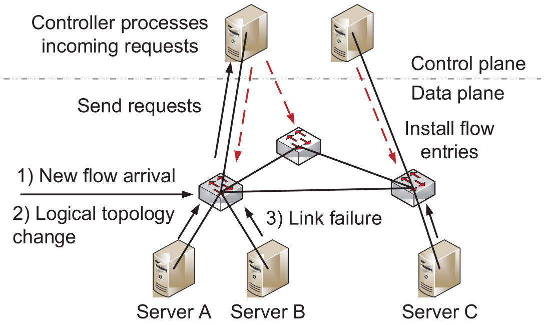


Fig. 1. The DCN model with SDN deployment. The requests sent by switches to controllers come from: 1) new flow arrivals, 2) logical topology changes [27], 3) link failures [35]. Controllers work both reactively and proactively to handle these requests.

II. Model and Formulation

In this section, we introduce the **D**ynamic **C**ontroller **A**ssignment **P**roblem (hereafter denoted as DCAP) along with the system model. We focus on understanding how to dynamically provision the control plane's capacity (i.e., the number of active controllers) and decide the assignment between switches and controllers so as to minimize the total cost in the data center deployed with SDN.

A. Network Model

Though the physical topology of a DCN varies [5], [19], 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. Note the SDN controllers can be running on dedicated hardware or software appliances in virtual machines [28], [48].

The network data plane has N switches, denoted as the set $\mathcal{S} = \{s_1, s_2, \cdots, s_N\}$, connected with end hosts. In the control plane, we have a set of preconfigured controllers, denoted as $\mathcal{C}$ with size $|\mathcal{C}|$. Their processing capacities are denoted as $\alpha = \left\{\alpha_1, \alpha_2, \cdots, \alpha_{|\mathcal{C}|}\right\}$ in terms of the number of requests it can handle in one time unit. Also, to handle the bursty traffic in DCN, there is a decay factor for each controller (denoted as $\left\{\beta_1, \beta_2, \cdots, \beta_{|\mathcal{C}|}\right\}, \beta_j \in \ (0,1), j = 1, 2, \cdots, |\mathcal{C}|$) to model the spare capacity. In time slot t, choosing from the total set of controllers $\mathcal{C}$, the control plane consists of $m(t)$ active controllers, denoted as $\mathcal{C}(t) = \left\{c_1, c_2, \cdots, c_{m(t)}\right\}$. Table I summarizes the key notations.

The assignment between switches and controllers is denoted as a binary $N \times m(t)$ matrix $\mathcal{X}(t)$, where

$$x_{ij}(t) = \begin{cases} 1, & i^{\text{th}} \text{ switch is connected to } j^{\text{th}} \text{ controller;} \\ 0, & \text{otherwise.} \end{cases}$$

To satisfy the liveness constraint [16], a switch must be connected to exactly one controller as its master. Hence, the sum of the elements in a row of matrix $\mathcal{X}(t)$ is equal to 1 (i.e., $\sum_{j=1}^{m(t)} x_{ij}(t) = 1$).

B. Controller Response Time Model

Since today's data center topology can provide high bisection bandwidth [5], [19], the propagation delay (in μs)

TABLE I
KEY NOTATIONS

Symbol	Semantics
$\mathcal{S}$	the set of switches, with size $\lvert\mathcal{S}\rvert = N$
s_i	i^{th} switch
$\mathcal{C}$	the complete set of controllers, with size $\lvert\mathcal{C}\rvert$
$\mathcal{C}(t)$	the set of active controllers in time slot t
c_j	j^{th} controller
α_j	processing capacity of j^{th} controller
β_j	decay factor of j^{th} controller
$\lambda_i(t)$	request rate of i^{th} switch in time slot t
d_{ij}	the hop distance between i^{th} switch and j^{th} controller
$m(t)$	the number of active controllers in time slot t
$x_{ij}(t)$	whether i^{th} switch is connected to j^{th} controller in time slot t

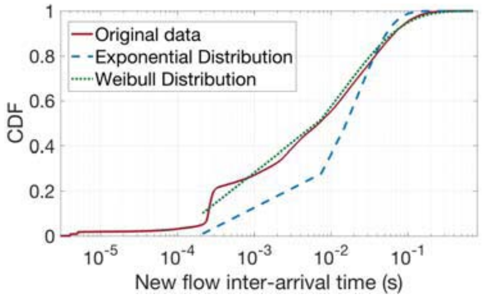


Fig. 2. CDF of flow inter-arrival times in the UNI1 DCN traffic dataset [1] publicly released by [10]. Best-fit curves for Exponential and Weibull distributions are depicted.

in dispatching forwarding rules is less significant than the controller CPU processing time (in ms) [16]. Thus we only model the request processing time on the controller.

We consider a discrete-time model where the length of a time slot matches the timescale at which switch requests can be recorded and the controller provisioning decisions be updated. The time slot may be 10 minutes during which the traffic pattern can be estimated [38]. There is an interval of interest $t \in \{1, 2, \cdots, T\}$. We now examine the flow inter-arrival time distributions in real-world DCNs. We filter unique SYN packets from the UNV1 dataset available in [1] and plot the CDF of inter-arrival times in Fig. 2. Observe that the CDF fits a Weibull distribution best; it also fits an exponential distribution reasonable well though sub-optimal than Weibull, which is consistent with [10]. Recent DCN traffic measurement [38] reports similar observations.

Though not ideally fit, we assume request arrivals to the i^{th} switch in slot t follows a Poisson process with parameter $\lambda_i(t) < \alpha_j,\ \forall\ j$. This is clearly a simplifying assumption. Traffic characteristics in production data centers are very complex and can be affected by factors such as topology, applications, etc. [26], [38], which make theoretical analysis intractable. Thus it is common in the literature to assume that network flow arrivals follow a Poisson process [7], [8], [21], [47]. Considering the requests to the controllers are directly triggered by the new flow arrivals and hence the requests are subject to the new flow arrivals, we assume that the request arrival process is also Poisson.

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

$$\theta_j(t) = \sum_{i=1}^{N} \lambda_i(t) x_{ij}(t). \tag{1}$$

We make the following assumptions: (1) in each time slot t, $\lambda_i(t)$ are mutually independent across i. (2) A controller can be modeled as an M/M/1 queue. By applying the Little's law, the average sojourn time is $\frac{1}{\alpha_j - \theta_j(t)}$. Given that the time of computing single source route is subject to the network size [15], the average processing time of the j^{th} controller can be calculated as below:

$$\vartheta_j(t) = \frac{1}{\alpha_j - \theta_j(t)} \cdot O(|\mathcal{S}|^2), \tag{2}$$

where $|\mathcal{S}|$ is the number of switches as in Table I.

The overall controller response time in t can be represented as the weighted average of $\{\vartheta_j(t), \forall j\}$:

$$\zeta(t) = \frac{\sum\limits_{j=1}^{m(t)} \theta_j(t)\vartheta_j(t)}{\sum\limits_{j=1}^{m(t)} \theta_j(t)}. \tag{3}$$

Rather than considering the maximum response time from the controllers (i.e., $\zeta(t)^* = \max\{\vartheta_j(t), \forall j\}$, using the weighted average takes into account each controller and their performance with its relative importances proportional to its load. Moreover, minimizing the weighted average also helps minimize the worst case response time as Theorem 1 shows.

Theorem 1: A control plane with lower weighted average response time also has lower worst case response time among all controllers.

The proof of Theorem 1 can be found in Appendix A in the supplementary material.

Further, to validate our controller response time model (i.e., Eq. (3)), we measure the response time of the *Ryu* controller [4] using the *cbench* tool [39]. On the testbed server that runs 64-bit CentOS 7 with an Intel E5620 2.4GHz CPU, we bind *Ryu* and *cbench* to different cores. We fix the number of fake switch to 1 and set other parameters to default values. We vary the request rate and trace the response time of each request. The measurements are carried out in 10 runs and the average response time is shown with error bars in Fig. 3.[1] As we can observe from Fig. 3, the average difference between our model and testbed measurement is ~0.0011ms,[2] which demonstrates the fidelity of our model.

C. Dynamic Controller Assignment Problem

Our objective is to decide the number of active controllers (i.e., $m(t)$) and the proper assignment between switches and controllers (i.e., $\mathcal{X}(t)$) to minimize the cost of the system, which can be divided into two categories:

[1] Since saturating the CPU leads to controller's performance degradation, we set the request rate low in our testbed.

[2] The capacity of the *Ryu* controller in our testbed is ~7500 requests per second.

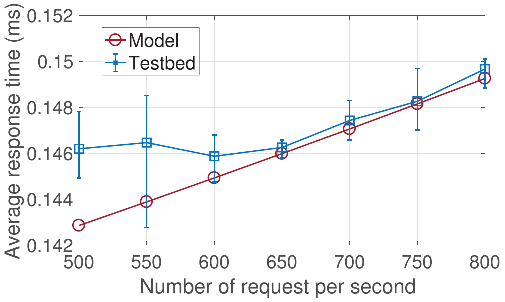


Fig. 3. The average controller response time of our model (i.e., Eq. (3)) and testbed experiment on Ryu controller [4].

- The *operating cost* incurred by the active controllers. It includes both the delay cost which depends on the control plane's response time (i.e., Eq. (3)) and the maintenance cost associated with the state synchronization.
- The *switching cost* incurred by turning controllers into and out of active state between each time slot, which may include the costs of machine setup, machine states migration, etc.

We now describe each type of cost in detail.

Operating Cost: The operating cost consists of two main components: (a) Delay cost: Compared with traditional networks, the deployment of SDN introduces additional processing time in the control plane. We define the total delay cost as:

$$\mathcal{D}(t) = f(\zeta(t)), \tag{4}$$

where $f(\cdot)$ is the cost function of the response time. The function $f(\cdot)$ can take any form to model a provider's specific delay cost factors. We only require that it is monotonically increasing in $\zeta(t)$.

(b) Maintenance cost: The large number of controllers complicates the management of the control plane. Particularly, to hold a consistent global view of the network, the controllers constantly communicates with each other to synchronize the network states [32]. We define this cost as:

$$\mathcal{M}(t) = g(m(t)), \tag{5}$$

where $g(\cdot)$ is the cost function of the state synchronization overhead. We require that $g(\cdot)$ is monotonically increasing and convex in $m(t)$. Typically, the controllers need to communicate with each other for states update, resulting in an all-to-all communication pattern [9], [32]. Thus, we define $g(\cdot)$ as a quadratic function, i.e., $g(m(t)) = \psi \cdot (m(t))^2$, where ψ is a parameter related to the cost of the inter-controller operations (e.g., communication cost, states update cost, etc.).

Switching Cost: We define δ as the unit cost of transitioning one controller from the sleep state to active state. The cost of transitioning from active to sleep is assumed to be zero. Thus, the switching cost for changing the number of active controllers from time slot $t-1$ to t is:

$$\mathcal{W}(t) = \delta \cdot [m(t) - m(t-1)]^+, \tag{6}$$

where $[\cdot]^+ = \max\{\cdot, 0\}$. The operator may set the unit cost δ according to the energy price [12], VM state migration costs [13], etc.

Given the workload and the cost models we have introduced above, we formally formulate the DCAP as a cost minimization problem. Specifically, our focus is to choose the assignment $\mathcal{X}(t)$ and the number of active controllers $m(t)$ at each time slot t in order to minimize the total cost during $[1, 2, \cdots, T]$. Thus, the DCAP can be expressed as:

$$\min \sum_{t=1}^{T} (\mathcal{D}(t) + \mathcal{M}(t) + \mathcal{W}(t)) \tag{7}$$

$$s.t.\ \theta_j(t) \le \beta_j \alpha_j, \quad \forall j,\ t, \tag{8}$$

$$\sum_{j=1}^{m(t)} x_{ij}(t) = 1, \quad \forall i,\ t, \tag{9}$$

$$x_{ij}(t) \in \{0, 1\}, \quad \forall i,\ j,\ t, \tag{10}$$

$$(1), (2), (3), (4), (6).$$

$$\text{var. } x_{ij}(t),\ m(t). \quad \forall i,\ j,\ t,$$

where the inequality (8) 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 (9) ensures that each switch is connected to exactly one master controller at the given time.

Discussion: The unit costs $f(\cdot), \psi, \delta$ in Eq. (4), (5), (6) serve also as normalization parameters to balance the relative importance of different types of costs according to their impact on revenue. We also comment that the one-time-slot version of DCAP (7) boils down to minimizing the control plane's response time (i.e., $\min \zeta(t)$). Unfortunately, the reduced one-time-slot optimization is a variant of the generalized assignment problem [14], which is NP-hard and also large scale in the context of DCN. Thus, we need to address the issue of computational efficiency when solving DCAP.

III. The Online Algorithm Framework

In this section, we present our online solution algorithm for DCAP. First we study DCAP in the offline setting, which involves a series of general assignment problems at each time slot that are NP-hard. We thus devise a two-phase algorithm to efficiently solve the switch-controller assignment problem in the offline setting. In the first phase, we transform the assignment problem to a one-to-many stable matching problem and obtain a solution which guarantees the worst-case response time. In the second phase we improve the matching solution by using a coalitional game approach that further reduces the response time. We also discuss the connection between the two phases, and advantages of using this method in optimizing the overall performance. Finally, drawing upon the two-phase offline algorithm and the classical Randomized Fixed Horizon Control framework [49], we design an online algorithm to solve the original DCAP.

A. Offline DCAP

We first study the DCAP in the offline setting, where the mean request arrival rate $\lambda_i(t)$ of the i^{th} switch over all time slots $[1, 2, \cdots, T]$ is known at the beginning of $t = 1$.

Given the arrival rates, essentially the offline DCAP has two facets: determining the number of active controllers $m(t)$, and the switch-controller assignment $\mathcal{X}(t)$ at each t. Also note that given $\{m(t)\}$ for all t, both the maintenance cost $\mathcal{M}(t)$ and the switching cost $\mathcal{W}(t)$—which only depends on the past state $m(t-1)$ and the current state $m(t)$—are fixed at every t. The offline DCAP with given $\{m(t)\}$ then boils down to:

$$\begin{aligned}&\min \sum_{t=1}^{T} \mathcal{D}(t)\\ &\text{s.t. } (1), (2), (3), (8), (9), (10),\\ &\text{var. } x_{ij}(t),\ \forall i,\ j,\ t.\end{aligned}$$

This is clearly equivalent to minimizing the delay cost $\mathcal{D}(t)$ at each t individually, that is,

$$\begin{aligned}&\min\ \mathcal{D}(t) \qquad (11)\\ &\text{s.t. } (1), (2), (3),\\ &\qquad \theta_j(t) \le \beta_j \alpha_j, \quad \forall j, \quad \sum_{j=1}^{m(t)} x_{ij}(t) = 1,\ \forall i,\\ &\text{var. } x_{ij}(t), \quad \forall i,\ j.\end{aligned}$$

Thus if problem (11) can be solved efficiently, the offline DCAP can be solved as a shortest path problem. Specifically, construct a directed graph G as follows. There are T stages. Each stage $t = [1, 2, \ldots, T]$ has exactly $|\mathcal{C}|$ nodes denoted as v_k^t where $|\mathcal{C}|$ is the total number of controllers in the network. Each node in stage $t-1$ has a directed link to each node in the next stage t, and no links to other nodes in the graph. There is a single source node connecting to each node at stage 1. The weight of the link between v_k^{t-1} and $v_{k'}^t$ is the minimum delay cost $\mathcal{D}^*(t)$ plus the maintenance cost $\mathcal{M}(t)$ and the switching cost $\mathcal{W}(t)$ given $m(t-1) = k$ and $m(t) = k'$. The initial and terminal nodes are denoted as v^0 and v^{T+1}, respectively. For $t = 0$, $m(0)$ is given as the initial state and the weights of its links can also be similarly obtained. It can be seen that the offline DCAP is equivalent to finding a shortest path on this graph G with the above setup, which can be solved in polynomial time using the Dijkstra's algorithm [15].

However, the switch-controller assignment problem at each time slot is in fact a generalized assignment problem which is NP-hard [14]. The problem itself is also large-scale as discussed in Sec. II-C, and needs to re-computed whenever the network conditions change. Thus in the following, we turn to focus on developing computationally efficient algorithms for switch-controller assignment. Traditional methods [9] for solving large-scale optimization are computationally prohibitive. We instead resort to stable matching and game theory techniques which have been regarded as efficient alternatives for tackling networking problems [45].

B. The Stable Matching With Transfers

To formally study the problem (11), we use the stable matching framework [17] with the classical deferred acceptance algorithm (DAA) [36]. 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 problem (11) as a many-to-one stable matching problem. Conventionally, we take the notation that $a \succ_c b$ means c prefers a to b.

1) The Stable Matching Phase:

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 other switches connected to it.

Most previous work on many-to-one stable matching [17] assumes that the preferences are static and independent of other members. As discussed in [37], 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 $\theta_j(t)$ in Eq. (2):

$$\vartheta_j^{max}(t) = \frac{O(|\mathcal{S}|^2)}{\alpha_j - \beta_j \cdot \alpha_j}. \qquad (12)$$

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. (12).

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.

Controller's 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., $\theta_j(t) + \lambda_{i^*}(t) \le \beta_j \cdot \alpha_j, \forall s_{i^*}$. 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 $\lambda_{i^*}(t) \cdot d_{i^*j}$.

However, differently from the traditional college admissions problem [17], our DCAP 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 does not directly related to the number of switches it can serve, the controller's quota depends on which switches it is assigned to.

Note that the switch's objective is to seek for lower response time. Thus intuitively, in a matching Θ (i.e., $\Theta(s_i) = c_j$ means s_i is assigned to c_j), if another controller c_{j^*} can provide the switch s_i with lower response time compared with s_i's currently assigned controller c_j, then the switch s_i has the incentive to break up from its current matching (i.e., $\Theta(s_i) = c_j \rightarrow \Theta(s_i) = c_{j^*}$). This rematching can happen when one of the followings satisfies: (1) c_{j^*} has the vacant capacity; (2) by rejecting some lower ranked connected switches, c_{j^*} has the vacant capacity. 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 $\theta_{j^*}(t) + \lambda_i(t) \leq \alpha_{j^*}\beta_{j^*}$.

(2) $c_{j^*} \succ_{s_i} \Theta(s_i)$, and $\theta_{j^*}(t) - \sum_{i^*} \lambda_{i^*}(t) + \lambda_i(t) \leq \alpha_{j^*}\beta_{j^*}$, 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 [17], 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 use the DAA [36] 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.

Theorem 2: Algorithm 1 yields a stable matching solution.

The proof of Theorem 2 can be found in Appendix B in the supplementary material.

Algorithm 1 Stable Matching Phase Procedure

```
Input: Request rate of each switch during time slot $t$: $\lambda_i(t)$
    Processing capacity of each controller: $\forall j$, $\alpha_j$
    Decay factor of each controller: $\forall j$, $\beta_j$
Output: Mapping between switches and controllers: $x_{ij}(t)$, $\forall i, j$
function StableMatching(measured request arrival rate $\forall i$, $\lambda_i(t)$, controller capacity $\forall j$, $\alpha_j$, decay factor $\forall j$, $\beta_j$)
    Each switch builds its own preference list according to the worst response time presented in Eq. (12): $\forall i$, $\Gamma(s_i)$
    Each controller builds its own preference list according to the additional traffic overhead defined in Definition 2: $\forall j$, $\Gamma(c_j)$
    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. (8) 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 $\Theta$ to $x_{ij}(t)$, $\forall i, j$.
end function
```

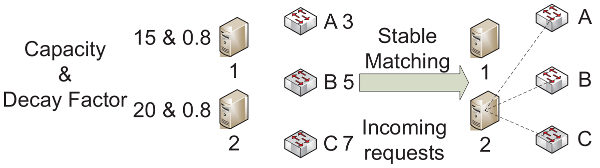


Fig. 4. The distance between switches and controllers is 1 hop. The stable matching solution yields a unbalanced situation that can be further improved by a transfer pair $\langle switch_B \rightarrow controller_1 \rangle$ considering response time.

2) Combination of Stable Matching and Coalitional Game: The stable matching framework is efficient and practical to tackle large-scale problems [45], [46]. However, solely applying stable matching may produce an unbalanced matching. In the stable matching shown in the Fig. 4, all switches are connected to controller-2 leaving controller-1 completely idle.

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. IV. 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 other words, 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 from 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 mitigate the unbalanced phenomenon.

3) 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

Algorithm 2 Coalitional Game Phase Procedure

```
Input: Partition $\mathcal{P} = \{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_{|\mathcal{C}|}\}$ obtained from Algorithm 1
    Processing capacity of each controller: $\forall j,\ \alpha_j$
    Decay factor of each controller: $\forall j,\ \beta_j$
Output: Mapping between switches and controllers: $x_{ij}(t)$, $\forall i, j$
function CoalitionalFormation(initial partition $\mathcal{P}$, controller capacity $\forall j,\ \alpha_j$, decay factor $\forall j,\ \beta_j$)
    repeat
        Each switch computes its most preferred transfer and proposes to the corresponding controller.
        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
```

disjoint partitions that are connected to different controllers. This phase can be seen as a coalition formation game [22] 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 matching Θ^* with 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 (8) of controller b^*.

(2) As defined in Eq. (2), transfer value satisfies $TV(s, a, b) = \zeta^{\Theta^*}(t) - \zeta^{\Theta}(t) < 0$.

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 as indicated in Eq. (3) 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: Once Algorithm 1 terminates with a stable matching Θ, the corresponding partition $\mathcal{P} = \{\mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_{|\mathcal{C}|}\}$ over $\mathcal{S}$ is used as an initial partition $\mathcal{P}_{initial}$ for the coalitional game in the second phase. Then, we iteratively find the transfer pair with minimum transfer value which means the lowest social welfare we can get as shown in Algorithm 2.

Theorem 3: Given $\mathcal{P}_{initial}$ obtained from Algorithm 1, Algorithm 2 converges to a Nash stable partition $\mathcal{P}_{final}$.

TABLE II
KEY SYMBOLS FOR THEOREM 4, 5 AND 6

Symbol	Semantics
ξ	processing capacity of a controller
$\overline{\theta(t)}$	average request rate on each controller in time slot t
$\varphi_j^{min}(t)$	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 $\overline{\theta(t)}$

Algorithm 3 Approximation Algorithm for DCAP in the Offline Setting

```
Input: Request rate of each switch: $\forall\ t, \lambda_i(t)$
    Processing capacity of each controller: $\forall j,\ \alpha_j = \xi$
    Decay factor of each controller: $\forall j,\ \beta_j$
    Cost function $f(\cdot)$, $g(\cdot)$ and constant parameter $\delta$
Output: Mapping between switches and controllers: $x_{ij}(t),\ \forall\ i, j, t$
    Number of active controllers: $m(t),\ \forall\ t$
    The total cost (7)
function OfflineDCAP($\lambda_i(t)$, $\forall j$, $\forall\ j, \alpha_j = \xi$, $\forall\ j, \beta_j$, $f(\cdot)$, $g(\cdot)$, $\delta$)
    for $t = 1, 2, \ldots, T$ do
        for $v_k^{t-1}$, $v_{k'}^t$, $\forall\ k$ do
            Calculate the weight of link between $v_k^{t-1}$ and $v_{k'}^t$ based on $\mathcal{D}(t)$ obtained by Algorithm 1 and 2.
        end for
    end for
    Find the shortest path from $v^0$ to $v^{T-1}$. The vertices of the shortest path are exactly the number of active controllers at each time slot $t$. And the sum of the edge weights along the shortest path is the total cost.
end function
```

We consider a simple situation where the capacity of controllers is identical (denoted as ξ), which can be achieved by using the same software configuration. Key notations are listed in the Table II. Then, we have the following theorem:

Theorem 4: The cost of the proposed hierarchically two-phase algorithm (i.e., Algorithm 1 and 2) is at most $\frac{f(1+\frac{\xi\cdot\varphi^{max}(t)}{\overline{\theta(t)}\cdot(\xi-\overline{\theta(t)}-\varphi^{max}(t))})}{f(1)}$ times the optimal cost in problem (11), where $\varphi^{max}(t) = \max\{\varphi_i^{min}(t), i \in \mathcal{C}^U\}$.

We give the proofs of Theorem 3 and 4 in the Appendix C and D, respectively, in the supplementary material.

Having solved the NP-hard problem (11) by leveraging the proposed two-phase algorithm (i.e., Algorithm 1 and 2), we are now able to calculate the aforementioned weight the link between v_k^{t-1} and $v_{k'}^t$ and present the algorithm to solve DCAP in the offline setting as shown in Algorithm 3.

Theorem 5: The approximation ratio of Algorithm 3 to the DCAP in the offline setting is $\frac{f(1+\frac{\xi\cdot\nu}{\kappa\cdot(\xi-\kappa-\nu)})}{f(1)}$, where $\nu = \max\{\varphi^{max}(t),\ \forall\ t\}$ and $\kappa = \min\{\overline{\theta(t)},\ \forall\ t\}$.

The proof of Theorem 5 is given in Appendix E in the supplementary material.

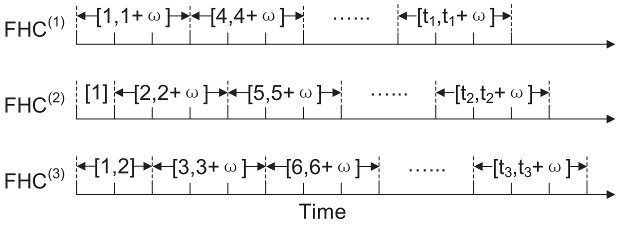


Fig. 5. An example of different FHC algorithms with $\omega = 2$, where $t_1 \in \Omega_1, t_2 \in \Omega_2, t_3 \in \Omega_3$.

C. Online Algorithm for DCAP

We have studied the DCAP in the offline setting, where in each time slot, we leverage stable matching with coalitional game techniques to solve the NP-hard problem (11). Now, we present an online algorithm by applying Randomized Fixed Horizon Control (RFHC) framework [49] and the offline algorithm in Sec. III-B.

Recent measurement studies on data center network show that traffic remains fairly stable in both long and short time intervals [38]. Further, traffic generated from applications, which follows some specific patterns (e.g., Directed Acyclic Graph and etc.), can be accurately predicted [40]. Therefore, we assume that the near-future information (i.e., the request arrival rate of each switch) can be estimated perfectly within the look-ahead window of ω time slots through a time series forecasting model [11]. The details of such a prediction module are beyond the scope of this work. In other words, at any time slot t, given the request rate of each switch until the time slot $t+\omega$, how can we fully utilize such information to make better decisions online?

We begin by introducing Fixed Horizon Control (FHC) briefly. Time is divided into equal-sized $\omega + 1$ time slots. At the beginning of time slot t, the request arrival rates for the next ω time slots, i.e., $\lambda_i(t+1), \lambda_i(t+2), \ldots, \lambda_i(t+\omega), \forall\ i$, are known. By using the Algorithm 3 proposed in Sec. III-B, we solve the following cost minimization problem over $t, t+1, \ldots, t+\omega$, given the decision of $m(t-1)$, to derive assignment decision $\mathcal{X}(\tau)$ and number of active controllers $m(\tau),\ \forall\ \tau = t, \ldots, t+\omega$:

$$\begin{aligned}
&\min \sum_{\tau=t}^{t+\omega} \mathcal{D}_{m(\tau)}(\tau) + \sum_{\tau=t}^{t+\omega} g(m(\tau)) + \delta[m(\tau) - m(\tau-1)]^{+} \\
&s.t.\ m(\tau) \in \{1, 2, \cdots, |\mathcal{C}|\}, \quad \forall \tau \\
&\qquad (1), (2), (3), (4), (6), (8), (9), (10). \\
&\text{var. } x_{ij}(\tau). \quad \forall i,\ j,\ \tau, \qquad (13)
\end{aligned}$$

Allowing the first time frame to start from different initial time slot $p \in [1, 1+\omega]$, we have $1+\omega$ versions of FHC algorithm, denoted as FHC$^{(p)}$. Taking Fig. 5 as an example with the size of look-ahead window being 2 (i.e., $\omega = 2$), we can start the algorithm at time slot 1, 2 or 3, resulting in a total of 3 versions of FHC algorithm. Thus the FHC$^{(p)}$ is solved at the times $\Omega_p = \{\tau | \tau \bmod(\omega+1) = p, \tau, p \in \mathbb{Z}, \tau \in [1, T], p \in [1, \omega+1]\}$.

The Randomized Fixed Horizon Control (RFHC) [49] used in Algorithm 4 adapts from FHC. We uniformly choose the start time $p \in [1, \omega+1]$ at random. Algorithm 4 iteratively solves problem (13) at the times Ω_p for the assignment strategy and controllers provisioning in the following $\omega+1$ time slots. Unlike FHC, RFHC eliminates choosing the small number of active controllers at the beginning of the time, which may incur large switching cost.

Algorithm 4 An Online Algorithm for DCAP

```
Input: Predicted request rate of each switch: $\forall\ t, \lambda_i(t)$
    Processing capacity of each controller: $\forall j,\ \alpha_j = \xi$
    Decay factor of each controller: $\forall j,\ \beta_j$
    Cost function $f(\cdot)$, $g(\cdot)$ and constant parameter $\delta$
    Look-ahead window $\omega$
Output: Mapping between switches and controllers: $x_{ij}(t),\ \forall\ i, j, t$
    Number of active controllers: $m(t),\ \forall\ t$
    The total cost (7)
function OnlineDCAP($\lambda_i(t)$, $\forall j$, $\forall\ j, \alpha_j = \xi$, $\forall\ j, \beta_j$, $f(\cdot)$, $g(\cdot)$, $\delta$)
    uniformly choose $p \in [1, 1+\omega]$ at random.
    if $p \neq 1$ then
        Derive $\mathcal{X}(\tau)$ and $m(\tau), \forall\ \tau = 1, 2, \ldots, p-1$ by solving problem (13) over time frame $[1, p-1]$ using Algorithm 3.
    end if
    Let $t = p$
    while $t \leq T$ do
        if $t \in \Omega_p$ then
            Derive $\mathcal{X}(\tau)$ and $m(\tau), \forall\ \tau = t, t+1, \ldots, t+p$ by solving problem (13) over time frame $[t, t+p]$ using Algorithm 3.
        end if
        $t = t + 1$.
    end while
end function
```

Theorem 6: . The competitive ratio of Algorithm 4 to the DCAP in the online setting is $\frac{f(1+\frac{\xi\cdot\nu}{\kappa\cdot(\xi-\kappa-\nu)})}{f(1)}(1+\frac{\delta}{(1+\omega)\cdot\psi})$, i.e., the cost obtained by Algorithm 4 is at most $\frac{f(1+\frac{\xi\cdot\nu}{\kappa\cdot(\xi-\kappa-\nu)})}{f(1)}(1+\frac{\delta}{(1+\omega)\cdot\psi})$ times the optimal cost in the offline setting.

The proof of Theorem 6 is given in Appendix F in the supplementary material.

D. Complexity Analysis

We now give a brief complexity analysis for algorithms presented above.

For Algorithm 1, as discussed in Sec. III-B, all the switches have the same preferences over controllers. Thus, Algorithm 1 terminates in $|\mathcal{C}|$ iterations. In each iteration, one controller accepts its most preferred switches. Sorting the switches needs $O(Nlog_2(N))$ computation. Therefore the time complexity of Algorithm 1 is $O(|\mathcal{C}|Nlog_2(N))$. For Algorithm 2, the complexity comes from searching for necessary transfers where, from the switches' perspective, the search space is $O(N|\mathcal{C}|)$ in each round. Thus, the hierarchical two-phase algorithm's computation complexity is $O(N|\mathcal{C}|) + O(N|\mathcal{C}|log_2(N)) = O(N|\mathcal{C}|log_2(N))$.

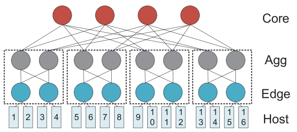


Fig. 6. 4-pod fat-tree (4 core, 8 aggregate and 8 edge switches, respectively) with controllers deployed on the hosts.

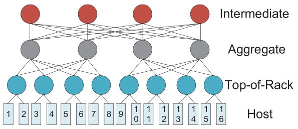


Fig. 7. VL2 ($D_I = 4$, $D_A = 8$, 4 intermediate, 4 aggregate, 8 ToR switches, respectively) with controllers deployed on the hosts.

For Algorithm 3, one-slot computation needs to calculate $|\mathcal{C}|$ different numbers of active controllers, resulting in $O(|\mathcal{C}|^2 Nlog_2(N))$ complexity. Since the length of the whole time frame is T and the complexity of calculating shortest path is $O((T|\mathcal{C}|)^2)$, the whole complexity of Algorithm 3 is $O(T|\mathcal{C}|^2 Nlog_2(N) + (T|\mathcal{C}|)^2)$.

For Algorithm 4, namely FHC^p, we blend the computation into two categories: 1) the previous $p-1$ time slots with $O((p-1)|\mathcal{C}|^2 Nlog_2(N) + ((p-1)|\mathcal{C}|)^2)$ complexity, 2) the other $\lfloor\frac{T-p}{1+\omega}\rfloor$ equal-sized frames with size of $1+\omega$, resulting in $O(\lfloor\frac{T-p}{1+\omega}\rfloor(1+\omega)|\mathcal{C}|^2 Nlog_2(N) + (\lfloor\frac{T-p}{1+\omega}\rfloor(1+\omega)|\mathcal{C}|)^2)$ computation complexity.

Considering that $|\mathcal{C}|$, the number of preconfigured controllers, is far smaller than the number of switches N, our proposed online Algorithm 4 is computationally efficient. We also exam the convergence time of the proposed algorithms in Sec. IV.

IV. Evaluation and Analysis

In this section, we conduct trace-driven simulations to evaluate the performance of our proposed algorithms. Specifically, the hierarchically two-phase algorithm (i.e., Algorithm 1 and 2) is hereafter referred to as the Stable Matching with Transfers (SMT) algorithm, and the online Algorithm 4 based on SMT is referred to as online SMT. We seek to answer the following: (a) What is SMT's performance on solving problem (11), which is NP-hard as discussed in Sec. III-A (Sec. IV-B)? (b) Why do we need a two-phase algorithm? Are the transfers necessary (Sec. IV-C)? (c) How does online SMT perform with limited future information (Sec. IV-D)?

A. Simulation Setup

Topology: We conduct our simulations using the widely adopted fat-tree [5] and VL2 [19] topologies as shown in Fig. 6 and Fig. 7, respectively. For fat-tree, the number of pods is 24 with a total of 3456 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 [10]. Hence, we set the $O(|\mathcal{S}|^2)$ as 10^4 in Eq. (2). Also, the capacity of each controller is 1800 k flows/s [16].

Trace: Since a request to the controller is directly triggered by a flow going through a switch, we set the request arrival rate in our simulations to follow the flow arrival rate distribution measured in a real-world data center [10]. We further introduce a load factor to scale down the request arrival rates, in order to evaluate our proposed algorithms in different load conditions because in practice, requests are not triggered by each flow arrival event.

Schemes Compared: **(1) SMT**: Our algorithm with two phases. The decay factor β_j is randomly chosen between 0.85 and 0.95. **(2) DCP-GK**: State-of-the-art algorithm from [9] 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 first derive the distance matrices for fat-tree and VL2, 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:

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

We conduct simulations of the following two settings. (A) *Single time slot setting*. This is to evaluate whether SMT can balance the load among controllers and reduce controller response time. In this setting, we run all algorithms in discrete time slots—which can be viewed as individual runs with different input requests—during which the request arrival rate changes according to its distribution over time. The number of controllers is fixed to 15, which is enough to handle the maximum request rate in our simulations.

(B) *Continuous time slot setting*. This is designed to evaluate whether the online SMT (i.e., Algorithm 4) can reduce the total cost. We compare SMT with online DCP-GK, which substitutes SMT in the online framework as we introduce in Sec. III-C, and the SM that is the static mapping between switches and controllers generated by DCP-GK in the first time slot. According to [2] where the cost of additional latency is reported, the function $f(\zeta(t))$ in the delay cost (4) is set as an linear function with parameter $\upsilon = 5e6$. And the ψ in maintenance cost $g(\cdot)$ and δ in switching cost (6) are set to

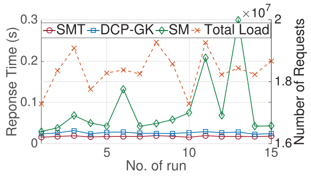


Fig. 8. In fat-tree, SMT reduces ∼33% controller response time on average, compared with DCP-GK.

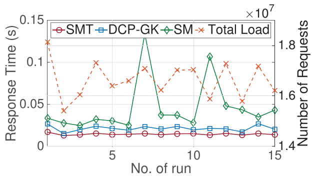


Fig. 9. In VL2, SMT reduces ∼31% controller response time on average, compared with DCP-GK.

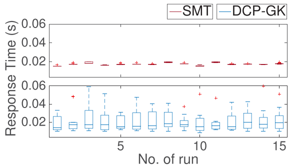


Fig. 10. Controller load distribution in fat-tree.

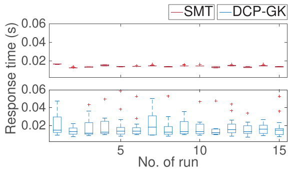


Fig. 11. Controller load distribution in VL2.

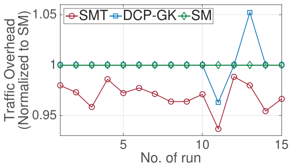


Fig. 12. Normalized traffic overhead in fat-tree.

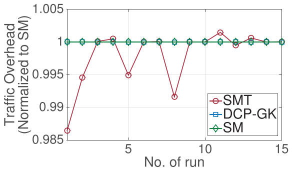


Fig. 13. Normalized traffic overhead in VL2.

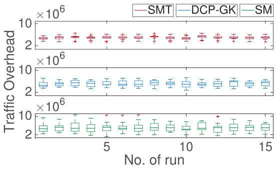


Fig. 14. Traffic overhead distribution in fat-tree.

0.1 and 1, respectively. Note that these 3 parameters can be the controlling knobs for operators the balances the mentioned costs. Unless otherwise stated, the look-ahead window ω is set to 2. The predictions in the look-ahead window are considered perfect without error. The experiments are carried out in continuous time slots in which the request arrival rate changes according to its distribution over time.

B. Effectiveness of SMT in Single Time Slot Setting

Response Time: In terms of controller response time in fat-tree and VL2, Fig. 8 and 9 plot the comparison of SMT, DCP-GK and SM in each individual run, where the number of requests varies as marked in the right-side y-axis. We make the following observations: (1) As the total number of requests 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. 8. In the extreme cases, the response time of SM is 17x that of SMT. It demonstrates that static controller assignment results in severe load imbalance. In our simulations such severe imbalance happens about 4% of the time in all runs. On average over all runs, SM incurs ∼4.7x and ∼3.1x controller response time compared with SMT in fat-tree and VL2, respectively. (3) SMT outperforms DCP-GK and reduces the response time by 32% on average (∼ 9ms), which has a significant impact on short flows' completion times in SDN the where control plane needs to quickly react to the events as discussed in Sec. I. Considering the stringent latency requirements of DCN [7], [8], a small delay can directly degrade user experience and hence hurt enterprise revenue. (4) Compared with the optimal response time, SMT is only 0.14% slower on average. To further understand the quantiles of the response time distribution, Fig. 10 and Fig. 11 depict the box-and-whiskers plots. It can be seen that the ratio of maximum and minimum response time of all controllers in DCP-GK is about 6.8 on average, while this ratio in SMT is only 1.2, which demonstrates SMT's ability to nearly equally distribute requests among controllers.

Control Traffic Overhead: Since each request needs to be sent to at least one controller [16], we denote the control traffic overhead as $\eta(t) = \sum_j^{m(t)} \sum_i^N d_{ij} x_{ij}(t) \lambda_i(t)$. Fig. 12 and Fig. 13 depict the control traffic overhead of the three algorithms, where all the overheads are normalized to SM's performance (approximately $7.69e7$). In Fig. 12, 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 3% less overhead compared with SM on average in Fig. 12. However, in Fig. 13, 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. Further, though the total control traffic overhead of DCP-GK and SM is nearly the same, the distributions are quite different as shown in the box-and-whiskers plots Fig. 14 and Fig. 15. It shows that the average ratio of maximum and minimum traffic overhead of all controllers in SMT, DCP-GK and SM is 1.7, 2.0 and 4.2, respectively.

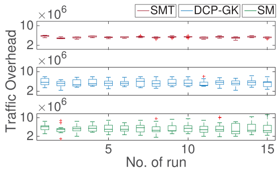


Fig. 15. Traffic overhead distribution in VL2.

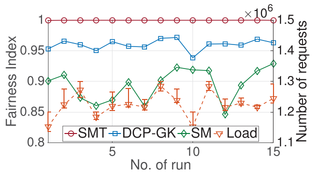


Fig. 16. Controller fairness index in fat-tree.

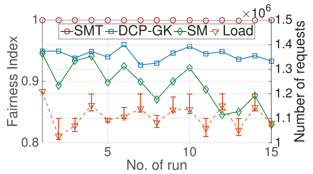


Fig. 17. Controller fairness index in VL2.

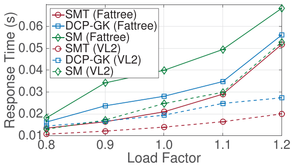


Fig. 18. Response times under different traffic loads.

C. Insights of SMT

Load Distribution of Controllers: To show the load distribution among multiple controllers along time, we plot the Jain's fairness index [24] (a value ranges from $\frac{1}{M}$ to 1) of these algorithms in Fig. 16 and Fig. 17. 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. 16 and Fig. 17 also plot the max/average/min loads on controllers in SMT of each individual run. It clearly shows that the load differential among multiple controllers is small, with the ratio of maximum and minimum load being 1.03 and 1.05, respectively.

Traffic Condition: As we have discussed in the experimental setting, switches do not always contact the controller for every flow arrival. Therefore, we introduce a load parameter to scale up/down the default request rate. In terms of response time, Fig. 18 shows the performance of all schemes under different traffic loads. It can be seen that response time of SMT increases slightly when loads increase, while SM increases sharply since the overload phenomenon becomes severer. This demonstrates SMT's ability to adapt to traffic variations in DCN.

Are Transfers Necessary?: One may ask why we need the second coalitional game phase and whether the transfers are necessary. Fig. 19 and Fig. 20 plot the difference between SMT and stable matching with no transfers (denoted as SMNT) in terms of response time and traffic overhead. For overhead, the transfers with transfer rule defined in Definition 5 bring about 4.1% more overhead to the network. However, they reduce the response time by about 57.6% compared with SMNT. This validates transfers' effectiveness to further reduce response time while incurring little more overhead to the network.

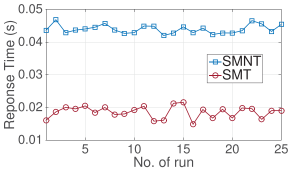


Fig. 19. Response time comparison between SMT and SMNT in fat-tree.

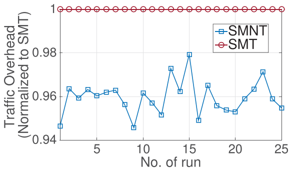


Fig. 20. Traffic overhead comparison between SMT and SMNT in fat-tree.

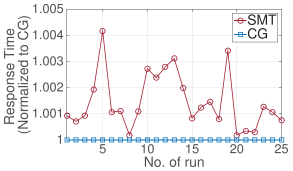


Fig. 21. Response time comparison between SMT and CG in fat-tree.

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. 21 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. In Fig. 21, we do find that the SMT experiences longer response time compared with CG, with only 1.5 ms difference on average. In terms of control traffic overhead, Fig. 22 shows that SMT incurs up to 6% 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. 23. 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.

D. Effectiveness of Online SMT

We now investigate how our SMT-based online Algorithm 4 performs. In the experiments below, we run Algorithm 4 in a continuous time frame which consists of 12 time slots.

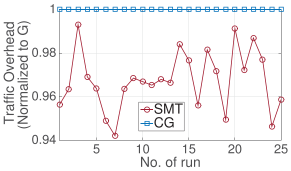


Fig. 22. Traffic overhead comparison between SMT and CG in fat-tree.

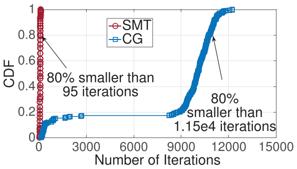


Fig. 23. The CDF of number of iterations in SMT and CG, respectively.

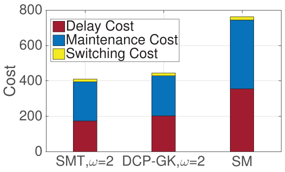


Fig. 24. Cost of online SMT, online DCP-GK and SM in fat-tree.

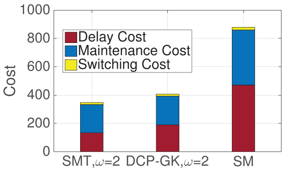


Fig. 25. Cost of online SMT, online DCP-GK and SM in VL2.

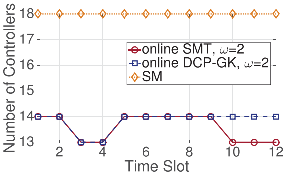


Fig. 26. Numbers of controllers allocated in each time slot of different algorithms in fat-tree.

Total Cost: As shown in Figs. 24 and 25, online SMT with a look-ahead window of 2 reduces the total cost by 8.4% and 46.2% compared with online DCP-GK and SM in fat-tree topology, while in VL2 topology, online SMT reduces the total cost by 14.7% and 64.2%, respectively. As the three types of cost—delay cost, maintenance cost and switching cost—is stacked in Figs. 24 and 25, we can observe that online SMT achieves the lowest delay cost. Compared with online DCP-GK, online SMT has nearly same maintenance cost, while has lower delay cost, which show SMT's ability to better load balance the controllers.

Number of Active Controllers: The number of active controllers during these 12 time slots is depicted in Fig. 26 and 27, it shows that provisioning the static number of controllers and remaining the static assignment between switches and controllers not only costs most but also has lowest performance considering response time (i.e., delay cost). Considering online SMT uses fewer active controller while achieves the lower cost compared with DCP-GK, it shows that our proposed algorithm can achieve a more balanced load distribution among controllers, which is consistent with our previous findings in Sec. IV-B.

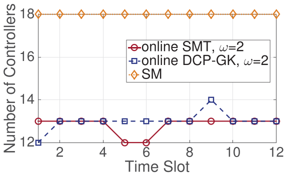


Fig. 27. Numbers of controllers allocated in each time slot of different algorithms in VL2.

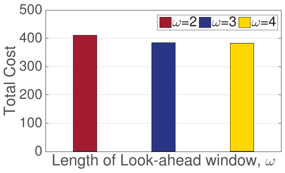


Fig. 28. Cost of online SMT with different length of look-ahead window in fat-tree.

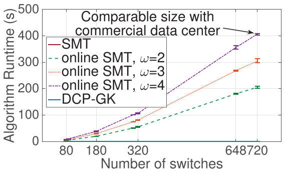


Fig. 29. Runtime of online SMT under different topology scales.

Look-Ahead Window, and Running Time: As discussed in Sec. III-C, the larger ω is, the more future information can be obtained, which can be leveraged to make better decisions towards lower cost. As depicted in Fig. 28, cost decreases by 4.8% and 7.2% when ω increases from 2 to 3 and 4, respectively.

Also, we examine the scalability of SMT. We set the number of pods in fat-tree to 24, 16, and 8 with 720, 320, and 80 switches, respectively, and set the degree of aggregate switches in VL2 to 48, 32, 24, and 16 with 648, 304, 180, and 88 switches, respectively. We use a server with Intel Xeon E5620 2.4GHz CPU. The measurements are carried out with 10 independent runs. Results are shown with error bars in Fig. 29. It can be observed that in the settings with a network size comparable to a commercial data center, SMT finishes in 0.2s, while greedy algorithm DCP-GK can finish in 0.1ms. As the look-ahead window ω increases, the runtime of online SMT increases slightly. The online SMT with $\omega = 2$ is able to finish in 200s, which fits the length of time slot (10 minutes). This demonstrates our proposed online SMT's ability to adapt to large-scale network.

V. Related Work

In this section, we survey the state of the art of SDN controller assignment, the use of stable matching and coalitional game in computer networking, and the online algorithms which considers look-ahead information.

Dynamic Controller Assignment

To improve robustness and scalability, several works [28], [48] 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 [3], Dixit *et al.* [16] 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.* [9] 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.* [29] 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 [16].

Our previous work [41] aims at mitigating the load imbalance among controllers and reducing response time in the one-time-slot setting. Here in this paper we introduce the maintenance cost of the controller cluster and formulate the controller assignment problem as a long-term online cost minimization.

Stable Matching and Coalitional Game

Stable matching is first introduced by Gale *et al.* [17] 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.* [45] 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 [44] to improve performance of VMs and reduce load imbalance on servers. They further extends the classical matching theory for general resource management in cloud computing with VM size heterogeneity in [46].

Game theory is one of the most widely used techniques in networking community. Han *et al.* [22] 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 one-time-slot assignment problem as a stable matching problem with transfers.

Online Algorithm With Look-Ahead Window

With advances in prediction models [11], near-future information can be precisely estimated. Receding Horizon Control (RHC) [31] is a classical control policy that incorporates the near-future information. Assuming look-ahead information can be obtained, several RHC-based online algorithms [30], [43] are proposed to dynamically provision capacity in data center. Addressing the degraded performance of RHC in heterogeneous environment, Lin *et al.* [33] propose Averaging Fixed Horizon Control (AFHC) to solve the load balancing problem among geographically-distributed data centers, which the competitive ratio has been shown to be significantly reduced with near-future information. To handle the integer constraints in our problem, instead of applying RHC or ARHC, we choose the Randomized Fixed Horizon Control (RFHC) framework [49] which has a small competitive ratio when applied to our work.

VI. Conclusion

In this paper, we studied the dynamic controller assignment problem (DCAP) as a online cost minimization. By applying Randomized Fixed Horizon Control (RFHC) framework, the online DCAP is decomposed to a series of one-time-slot assignment problem, which is cast as a stable matching problem with transfers to minimize the controller response time. The one-time-slot 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 and the overall performance of the proposed online algorithm guarantees a small loss in competitive ratio. 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. In the single time slot setting, the two-phase algorithm, which achieves near-optimal load balancing among controllers, reduces the controller response time by about 32% and 65% compared with state of the art DCP-GK and the simple static assignment, respectively. In the online setting, our online algorithm reduce the total cost by about 46% and 64% compared with static assignment in fat-tree and VL2 topologies, respectively.

References

[1] *Data Center Measurement Data Set*, accessed on Nov. 1, 2010. [Online]. Available: http://pages.cs.wisc.edu/~tbenson/IMC10_Data.html

[2] *Latency is Everywhere and it Costs Your Sales*, accessed on Jul. 25, 2009. [Online]. Available: http://highscalability.com/latency-everywhere-and-it-costs-you-sales-how-crush-it/

[3] *Openflow Specification v1.3*, Open Netw. Foundation, Menlo Park, CA, USA, 2012.

[4] *The Ryu Controller*, accessed on Jun. 25, 2012. [Online]. Available: https://osrg.github.io/ryu/

[5] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM*, 2008, pp. 63–74.

[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. USENIX NSDI*, 2010, p. 19.

[7] M. Alizadeh *et al.*, "pFabric: Minimal near-optimal datacenter transport," in *Proc. ACM SIGCOMM*, 2013, pp. 435–446.

[8] W. Bai *et al.*, "PIAS: Practical information-agnostic flow scheduling for data center networks," in *Proc. USENIX NSDI*, 2015, p. 25.

[9] M. F. Bari *et al.*, "Dynamic controller provisioning in software defined networks," in *Proc. IEEE CNSM*, Oct. 2013, pp. 18–25.

[10] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. ACM IMC*, 2010, pp. 267–280.

[11] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*. Hoboken, NJ, USA: Wiley, 2015.

[12] G. Chen *et al.*, "Energy-aware server provisioning and load dispatching for connection-intensive Internet services," in *Proc. USENIX NSDI*, 2008, pp. 337–350.

[13] C. Clark *et al.*, "Live migration of virtual machines," in *Proc. USENIX NSDI*, 2005, pp. 273–286.

[14] R. Cohen, L. Katzir, and D. Raz, "An efficient approximation for the generalized assignment problem," *Inf. Process. Lett.*, vol. 100, no. 4, pp. 162–166, 2006.

[15] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische. Math.*, 1959, pp. 269–279.

[16] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," in *Proc. ACM HotSDN*, 2013, pp. 7–12.

[17] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *Amer. Math. Monthly*, vol. 69, no. 1, pp. 9–15, Jan. 1962.

[18] R. Govindan, I. Minei, M. Kallahalla, B. Koley, and A. Vahdat, "Evolve or die: High-availability design principles drawn from googles network infrastructure," in *Proc. ACM SIGCOMM*, New York, NY, USA, 2016, pp. 58–72.

[19] A. Greenberg *et al.*, "VL2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM*, 2009, pp. 51–62.

[20] J. Guo, F. Liu, J. C. S. Lui, and H. Jin, "Fair network bandwidth allocation in IaaS datacenters via a cooperative game approach," *IEEE/ACM Trans. Netw.*, vol. 24, no. 2, pp. 873–886, Apr. 2016.

[21] J. Guo, F. Liu, T. Wang, and J. C. S. Lui, "Pricing intra-datacenter networks with over-committed bandwidth guarantee," in *Proc. USENIX ATC*, 2017.

[22] Z. Han, *Game Theory in Wireless and Communication Networks: Theory, Models, and Applications*. Cambridge, U.K.: Cambridge Univ. Press, 2012.

[23] B. Heller *et al.*, "ElasticTree: Saving energy in data center networks," in *Proc. USENIX NSDI*, 2010, pp. 249–264.

[24] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe, *A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems*, vol. 38. Hudson, MA, USA: Eastern Res. Lab., Digit. Equip. Corp., 1984.

[25] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," in *Proc. ACM SIGCOMM*, 2013, pp. 3–14.

[26] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *Proc. ACM IMC*, 2009, pp. 202–208.

[27] T. Koponen *et al.*, "Network virtualization in multi-tenant datacenters," in *Proc. USENIX NSDI*, 2014, pp. 203–206.

[28] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *Proc. USENIX OSDI*, 2010, pp. 1–6.

[29] A. Krishnamurthy, S. P. Chandrabose, and A. Gember-Jacobson, "Pratyaastha: An efficient elastic distributed SDN control plane," in *Proc. ACM HotSDN*, 2014, pp. 133–138.

[30] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," in *Proc. IEEE ICAC*, Jun. 2008, pp. 3–12.

[31] W. Kwon and A. Pearson, "A modified quadratic cost problem and feedback stabilization of a linear system," *IEEE Trans. Autom. Control*, vol. AC-22, no. 5, pp. 838–842, Oct. 1977.

[32] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: State distribution trade-offs in software defined networks," in *Proc. ACM HotSDN*, 2012, pp. 1–6.

[33] M. Lin, Z. Liu, A. Wierman, and L. L. Andrew, "Online algorithms for geographical load balancing," in *Proc. IEEE IGCC*, Jun. 2012, pp. 1–10.

[34] F. Liu, J. Guo, X. Huang, and J. C. S. Lui, "eBA: Efficient bandwidth guarantee under traffic variability in datacenters," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 506–519, Feb. 2017.

[35] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic engineering with forward fault correction," in *Proc. ACM SIGCOMM*, 2014, pp. 527–538.

[36] A. E. Roth, "Deferred acceptance algorithms: History, theory, practice, and open questions," *Int. J. Game Theory*, vol. 36, no. 3, pp. 537–569, 2008.

[37] A. E. Roth and M. A. O. Sotomayor, *Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis*. Cambridge, U.K.: Cambridge Univ. Press, 1992.

[38] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proc. ACM SIGCOMM*, 2015, pp. 123–137.

[39] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc. USENIX HotICE*, 2012, pp. 1–6.

[40] H. Wang *et al.*, "FLOWPROPHET: Generic and accurate traffic prediction for data-parallel cluster computing," in *Proc. IEEE ICDCS*, Jun. 2015, pp. 349–358.

[41] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic SDN controller assignment in data center networks: Stable matching with transfers," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[42] T. Wang, H. Xu, and F. Liu, "Multi-resource load balancing for virtual network functions," in *Proc. IEEE ICDCS*, 2017.

[43] X. Wang and M. Chen, "Cluster-level feedback power control for performance optimization," in *Proc. IEEE HPCA*, 2008, pp. 101–110.

[44] H. Xu and B. Li, "Egalitarian stable matching for VM migration in cloud computing," in *Proc. IEEE INFOCOM Workshop Cloud Comput.*, Apr. 2011, pp. 631–636.

[45] H. Xu and B. Li, "Seen as stable marriages," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 586–590.

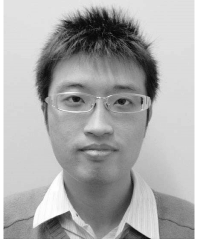
[46] H. Xu and B. Li, "Anchor: A versatile and efficient framework for resource management in the cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1066–1076, Jun. 2013.

[47] H. Xu and B. Li, "RepFlow: Minimizing flow completion times with replicated flows in data centers," in *Proc. IEEE INFOCOM*, May 2014, pp. 1581–1589.

[48] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," in *Proc. ACM SIGCOMM*, 2010, pp. 351–362.

[49] L. Zhang *et al.*, "Moving big data to the cloud: An online cost-minimizing approach," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 12, pp. 2710–2721, Dec. 2013.

Tao Wang received the B.Eng. degree from the School of Computer Science and Technology, Huazhong University of Science and Technology, China, where he is currently pursuing the M.Eng. degree. His research interests include data center networking and software-defined networking.



Fangming Liu (S'08–M'11–SM'16) received the B.Eng. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2005, and the Ph.D. degree in computer science and engineering from The Hong Kong University of Science and Technology, Hong Kong, in 2011. He was a StarTrack Visiting Faculty with Microsoft Research Asia from 2012 to 2013, and a Visiting Scholar with the University of Toronto from 2009 to 2010. He is currently a Full Professor with the Huazhong University of Science and Technology, Wuhan, China. His research interests include cloud computing and datacenter, mobile cloud, green computing, SDN/NFV, and virtualization. He received the National Natural Science Fund (NSFC) for Excellent Young Scholars, and the National Program Special Support for Top-Notch Young Professionals.

Hong Xu (S'08–M'13) received the B.Eng. degree from the Department of Information Engineering, The Chinese University of Hong Kong, in 2007, and the M.A.Sc. and Ph.D. degrees from the Department of Electrical and Computer Engineering, University of Toronto, in 2013. He joined the Department of Computer Science, City University of Hong Kong, in 2013, where he is currently an Assistant Professor. His research interests include data center networking, NFV, and cloud computing. He was a recipient of an Early Career Scheme Grant from the Research Grants Council of the Hong Kong SAR, 2014. He also received the best paper awards from the IEEE ICNP 2015.