Predictive Switch-Controller Association and Control Devolution for SDN Systems

Xi Huang, *Member, IEEE*, Simeng Bian, *Student Member, IEEE*, Ziyu Shao®, *Senior Member, IEEE*, and Hong Xu®, *Senior Member, IEEE, Member, ACM*

Abstract—For software-defined networking (SDN) systems, to enhance the scalability and reliability of control plane, existing solutions adopt either multi-controller design with static switch-controller association, or static control devolution by delegating certain request processing back to switches. Such solutions can fall short in face of temporal variations of request traffics, incurring considerable local computation costs on switches and their communication costs to controllers. So far, it still remains an open problem to develop a joint online scheme that conducts dynamic switch-controller association and dynamic control devolution. In addition, the fundamental benefits of predictive scheduling to SDN systems still remain unexplored. In this paper, we identify the non-trivial trade-off in such a joint design and formulate a stochastic network optimization problem which aims to minimize time-averaged total system costs and ensure long-term queue stability. By exploiting the unique problem structure, we devise a predictive online switch-controller association and control devolution (POSCAD) scheme, which solves the problem through a series of online distributed decision making. Theoretical analysis shows that without prediction, POSCAD can achieve near-optimal total system costs a tunable trade-off for queue stability. With prediction, POSCAD can achieve even better performance with shorter latencies. We conduct extensive simulations to evaluate POSCAD. Notably, with mild-value of future information, POSCAD incurs a significant reduction in request latencies, even when faced with prediction errors.

Index Terms—SDN, switch-controller association, control devolution, predictive scheduling.

I. INTRODUCTION

DURING the past decade, software-defined networking (SDN) has initiated a profound revolution in networking system design towards more efficient network management and more flexible network programmability. The key idea of

SDN is to separate the control plane from the data plane [27]. In such a way, the control plane maintains network-wide information to process requests that are constantly generated from SDN-enabled switches in the data plane. Based on the instructions from the control plane, the data plane only needs to carry out basic network functions such as monitoring and packet forwarding.

As the scale of data plane expands, scalability and reliability of control plane often become the main concerns for SDN systems [18]. For example, the control plane, if implemented with a singleton controller, can be overloaded by the ever-increasing request traffic, leading to excessively long queueing latencies and belated response to network events. Besides, the singleton controller is also a single point of failure which can result in the breakdown of the whole networking system.

To address such concerns, existing solutions basically fall into two categories. One is to implement the control plane as a distributed system with multiple controllers [3]. Such controllers cooperate to form a logically centralized control plane to manage the network within a single administrative domain [5], [17], or logically distributed control planes to handle networks across different domains [30], [32]. Under such designs, each switch may have potential connections to multiple controllers for fault tolerance and load balancing. Accordingly, the associations between switches and controllers need to be carefully determined so as to reduce the communication costs and balance the workloads among controllers. To this end, existing works [10], [11], [19], [20], [25], [36], [37] have proposed various solutions to decide *switch-controller associations* in a deterministic fashion. The other is to *devolve* the processing of some requests that requires no global information onto switches, to reduce workloads on the control plane [12], [44]. Such techniques have been widely considered and adopted in various large-scale systems such as data center networks [9], WAN [16], and edge computing [39]. For example, Curtis *et al.* [9] modified the OpenFlow model to conduct effective flow management by: 1) devolving the control of most flows back onto switches and processing them with an aggressive use of flow-match wildcards or hash-based routing, while 2) controllers maintain global visibility to handle only targeted significant flows; *e.g.*, carrying out load balancing for long-lived flows with high throughputs.

Based on such investigations, we identify several interesting but unresolved questions regarding the control plane design:

◇ Instead of conducting deterministic switch-controller associations with infrequent re-association [19], [36], can we directly perform dynamic association with respect to

request traffic variations? What is the benefit of fine-grained control at the request level?

⬦ How to conduct dynamic devolution?

⬦ Are there any trade-offs in the joint design of dynamic switch-controller association and dynamic control devolution? If so, how do we manage such trade-offs?

⬦ Since the uncertainty in request traffic statistics remains one of the key factors that bring challenges to SDN system design, then if they can be learned, what are the *fundamental* limits of the benefits of attaining such future information for SDN systems?

Notably, the last question is motivated by the recent growing interests in leveraging machine-learning-based predictive analytics and scheduling to improve system performance, such as traffic prediction for routing optimization [38], quality-of-experience (QoE) prediction to promote user satisfaction [7], and failure prediction to optimize IT operations [26]. Despite the proposal of various prediction-based approaches [8], [28] [42], [43] in recent years, the *fundamental* benefits of predictive scheduling for SDN systems remain unexplored.

In this paper, we focus on general SDN systems with requests dynamically generated from switches in the data plane for the processing of various network events. We assume that each request can be either processed at a switch (with computation costs) or be uploaded to certain controllers (with communication costs).[1] We aim to reduce the computational costs by control devolution at data plane, the communication costs by switch-user association between data plane and control plane, and the response times experienced by switches' requests, which is mainly caused by queueing delays on controllers. Regarding predictive scheduling, switches are assumed able to predict requests to arrive in a limited number of time slots ahead through lightweight prediction modules with recent time-series forecasting techniques [6]. Further, we assume such future requests can be generated and pre-served before their arrival, and, if mis-predicted, they will incur extra system costs of communication and computation.[2] Under such settings, we open up a new perspective to answer the above questions. We summarize our contributions as follows.

⬦ **Modeling and Formulation:** We formulate the problem stated above as a stochastic optimization problem that aims to minimize time-averaged total system costs with long-term queue stability constraints. Through a careful choice in the granularity of modeling and decision making, *i.e.*, to characterize system dynamics at request level and request scheduling on a time-slot basis, we achieve a decent balance between modeling accuracy and decision making complexity.

⬦ **Algorithm Design:** By adopting existing techniques [13], [29] and exploiting unique problem structure, we propose *POSCAD*, a Predictive Online Switch-Controller Association and control Devolution scheme which exploits

predicted request arrival information to make association and devolution decisions in a distributed fashion.

⬦ **Performance Analysis:** We conduct theoretical analysis which shows that without prediction, *POSCAD* yields a tunable trade-off between $O(1/V)$ deviation from minimum long-term average total costs of communication and local computation on switches, and $O(V)$ bound for long-term average queue backlog size. Furthermore, with prediction, *POSCAD* can achieve an even better performance with a notable reduction in request latencies, which is verified by our simulation results. Besides, we also discuss the insights of our design and implementation issues in practice.

⬦ **Performance Evaluation and Verification:**[3] We conduct extensive simulations to evaluate the performance of *POSCAD*. Our results show that under various settings, *POSCAD* achieves near-optimal system costs while maintaining a tunable trade-off with queue stability. Furthermore, given only mild-value of predicted information, *POSCAD* incurs a significant reduction in request response time, even when faced with mis-prediction.

⬦ **New Degree of Freedom in the Design Space of SDN Systems**: To the best of our knowledge, this paper provides the first design to explore and exploit the benefits of predictive scheduling for SDN systems, opening up a new perspective in the design of SDN systems.

The rest of this paper is organized as follows. Section II presents two motivating examples to illustrate the non-trivial trade-off between different performance metrics and the potential benefits of predictive scheduling in SDN systems, respectively. Section III demonstrates our system model and problem formulation. Then Section IV shows the design of *POSCAD* and its performance analysis. Section V discusses our simulation results, Section VI reviews related works, while Section VII concludes this paper.

II. MOTIVATING EXAMPLES

In this section, we provide two motivating examples to illustrate the non-trivial trade-off in the joint design of dynamic switch-controller association and control devolution, and the potential benefits of predictive scheduling in SDN systems, respectively. Such examples motivate our subsequent problem formulation of joint switch-controller association and control devolution with predictive scheduling in SDN systems.

A. Motivating Example of Performance Trade-Off

Fig. 1 shows the evolution of an SDN system within one time slot. Particularly, Fig. 1 (a) presents the initial system state at the beginning of the time slot. Fig. 1 (b) – Fig. 1 (e) show the system evolution given two different association decisions. In Fig. 1 (b), switches s_1 and s_2 are associated to controller c_1, whereas s_3 chooses to process its requests locally. In Fig. 1 (d), switches s_1 and s_3 are associated to controllers c_1 and c_2, respectively, while s_2 chooses to process its requests locally.

---

[1]The scenario in which some requests can only be processed by a controller is a special case of our model.

[2]The scenario in which some future requests may not be pre-served due to their dependency on the processing of previous results is also a special case of our model.

[3]Note that testbed-based experimental verifications are not considered in this work, since our focus is on the exploration of the *fundamental* limits of the benefits of predictive scheduling in SDN systems. Nonetheless, it can be an interesting direction for future work.
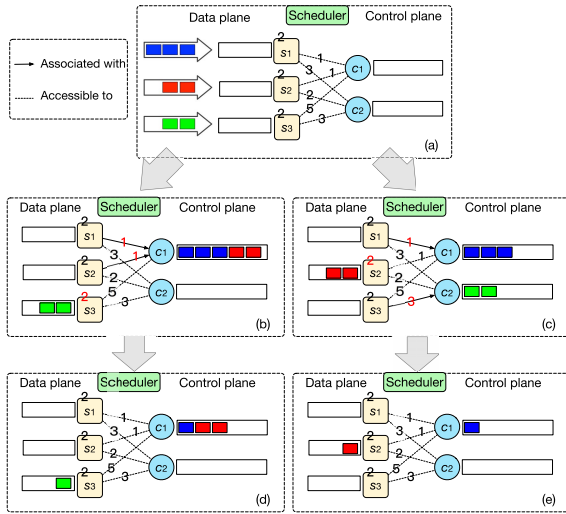
Fig. 1. An SDN system with request-level switch-controller association and control devolution within one time slot. There are three switches (s_1, s_2, s_3), two controllers (c_1, c_2), and one global scheduler. Switches and controllers maintain queue backlogs to buffer requests. Between each pair of controller and switch, each dotted line denotes a potential connection, while each solid line denotes the actual association made in the current time slot. Each connection and each switch are associated with a number denoting the unit communication or local computation costs (on switches) for transferring and processing one request, respectively. Each controller can serve up to 2 requests for each time slot, while each switch serves only 1 request. The scheduler's goal is to minimize the sum of total communication costs and computational costs, as well as the total queue backlog size, which implies timely processing of requests. The system proceeds as follows. At the beginning of time slot t, switches s_1, s_2, and s_3 generate 3, 2, and 2 requests, respectively. The scheduler then associates switches to controllers in two ways ((b) and (c)). Switches then process new requests locally or upload them to controllers. (a) System dynamics at the beginning of time slot t. (d) System dynamics in time slot t resulting from the scheduling in (b). (e) System dynamics in time slot t resulting from the scheduling in (c).

First, we focus on the consequences of different scheduling decisions for switch s_3. In Fig. 1 (b), switch s_3 chooses to process its requests locally, which incurs 2 units of computation costs per request. In Fig. 1 (c), s_3 decides to upload requests to c_2, which incurs 3 units of communication costs. Though the decision in Fig. 1 (b) leads to a lower total cost of 4 units than Fig. 1 (c) (of 6 units), it still leaves one request untreated at the end of the time slot.

Next, we switch to the consequences of such scheduling decisions to the whole system. In Fig. 1 (b), the decision (denoted by X_1) includes two switch-controller associations: (s_1, c_1) and (s_2, c_1) (s_3 processes requests locally). From Fig. 1 (d), we see that although decision X_1 only incurs total costs of 9, it still leaves four requests unfinished at the end of the time slot. Meanwhile, in Fig. 1 (c), the decision (denoted by X_2) includes two associations: (s_1, c_1) and (s_3, c_2) (s_2 processes its requests locally). Fig. 1 (e) show that X_2 does better than X_1 with two more finished requests at the end of the time slot. However, this is achieved at a higher total cost of 13 units.

The above observations show that there is a non-trivial trade-off between system costs and total queue backlog reduction.

B. Motivating Example of Benefits of Predictive Scheduling

Fig. 2 presents an example that compares the cases with and without predictive scheduling, where Fig. 2 (a) shows the
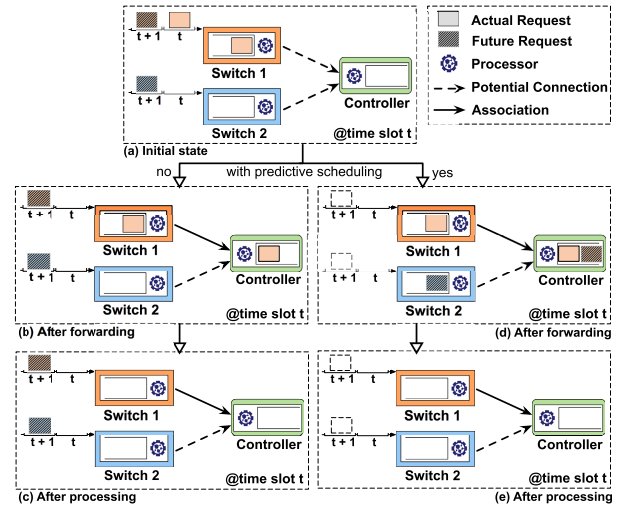

Fig. 2. An example that shows the benefits of predictive scheduling.

system state at the beginning of time slot t. There are two switches potentially connected to one controller (denoted by dashed arrows). All requests are assumed homogeneous such that they can be handled by both switches and the controller. Upon new requests' arrival, each switch either associates with the controller (denoted by a solid arrow) and uploads requests, or stores them in its own local processing queue. In each time slot, a switch processes at most 1 request and the controller processes at most 2 requests, both in a first-in-first-out manner. The objective is to minimize the average request response time. Fig. 2 (a) shows the system state at the beginning of time slot t, where one new request arrives at switch 1, one is already stored in switch 1's local queue, and no requests arrive at switch 2. The future request arrivals (marked with stripped colors) in time slot $(t + 1)$ are also visible to switches. Fig. 2 (b) and Fig. 2 (c) present the scheduling process that handles current requests only, whereas Fig. 2 (d) and Fig. 2 (e) exhibit the case with predictive scheduling.

First, we consider the case without predictive scheduling and handle only the arriving requests, as shown in Fig. 2 (b) and Fig. 2 (c). For switch 1, it chooses to make an association with the controller and uploads the new request, considering that its local queue has already buffered one request. Meanwhile, switch 2 takes no action since no requests arrive at present. After the only new request is forwarded, as shown in Fig. 2 (b), the controller and switches process the requests in their respective queues. Fig. 2 (c) shows the system state at the end of time slot t, where the two requests in time slot $(t + 1)$ are still left unprocessed.

Next, we focus on the case with predictive scheduling, as shown in Fig. 2 (d) and Fig. 2 (e). Switch 1 associates with the controller and uploads the new request that arrives at present. Considering that the controller has a service capacity of two requests per time slot, it pre-admits the request which will arrive in time slot $(t + 1)$, then uploads the request to the controller. Similarly, switch 2 pre-admits the future request in time slot $(t + 1)$ and stores the request in its local queue. Fig. 2 (d) shows the system state after the (pre-)admission of requests. Then the controller and switches

Symbol	Description
$\mathcal{C}$	The set of controllers in the control plane
$\mathcal{S}$	The set of switches in the data plane
$Q_i^s(t)$	Switch i's local queue backlog size in time slot t
$Q_j^c(t)$	Controller j's queue backlog in time slot t
$Q_i^p(t)$	Prediction queue backlog with respect to switch i in time slot t
$Q_i^{(d)}(t)$	The number of untreated requests by time slot t that will arrive at switch i in time slot $(t+d)$
$A_i(t)$	The number of request arrivals on switch i in time slot t
$B_i^s(t)$	The service capacity of switch i in time slot t
$B_j^c(t)$	The service capacity of controller j in time slot t
$M_{i,j}(t)$	The per-request communication cost between switch i and controller j in time slot t
$P_i(t)$	The per-request computational cost on switch i in time slot t
$X_{i,j}(t)$	The association decision with respect to switch i and controller j in time slot t
$Y_i(t)$	The admission decision for switch i in time slot t
$f_p(t)$	The total communication costs incurred in time slot t
$g_p(t)$	The total computational costs incurred in time slot t
$h_p(t)$	The weighted total queue backlog size in time slot t

consume the requests from their queues. Fig. 2 (e) shows that with predictive scheduling, all requests in time slots t and $(t+1)$ are completed by the end of time slot t. Consequently, both future requests will receive instant response upon their arrivals in time slot $(t+1)$.

The above example shows that predictive scheduling can effectively reduce the request response time by taking advantage of predicted future information.

III. PROBLEM FORMULATION

In this section, we present our system model and problem formulation, with key notations summarized in Table I.

A. System Model

1) Basic Model: We consider an SDN system that evolves over time slots, indexed by $t \in \{0, 1, 2, \ldots\}$. The system maintains a logically centralized control plane that comprises a set $\mathcal{C}$ of physically distributed controllers. Meanwhile, its data plane consists of a group of SDN-enabled switches $\mathcal{S}$. Each switch i keeps a local processing queue[4] of backlog size $Q_i^s(t)$, while each controller j maintains a queue backlog $Q_j^c(t)$ that buffers requests from the data plane.

In each time slot t, there are a number of $A_i(t)$ ($\leq a_{max}$ for some constant a_{max}) new requests arriving at each switch i.[5] Such arrivals $A_i(t)$ are assumed independently and identically distributed over time slots. We denote $\mathbf{A}(t) \triangleq \{A_i(t)\}_i$. There are two cases of serving a request. One case is local

---

[4]In some other designs [12], the control devolution can also be implemented with a two-layered control plane. Controllers in the bottom layer are deployed in the virtual machines on the same servers as some Open vSwitches. Such controllers have no network-wide information but they can provide faster processing of some basic functions such as local load balancing. Our model can be directly extended to handle such cases by regarding the local processing as the service provided by the controllers in the bottom layer.

[5]In our work, we define a request as follows. When a new packet arrives at an SDN-enabled switch, the switch will extract the packet's header fields to match against flow table entries and execute the matched action upon the packet. If the packet matches no entries (or matches some pre-specified entry), the switch will trigger a packet-in event, encapsulate the packet, then forward it to one of the controllers. In this process, we call the processing demand of each packet a *request*.

processing, which means that the switch is programmed to provide local control functions to process the packet with sub-optimal but faster decision making [9]. The other case is when the packet is added into an SDN event and uploaded to the control plane to decide how to update the flow table.

In this work, we assume all requests are homogeneous; *i.e.*, they can be handled by both switches and controllers, though our model can be directly extended to scenarios with heterogeneous requests with stateful processing requirements. To process such requests, each controller j has a service capacity of $B_j^c(t)$ requests, while each switch $i \in \mathcal{S}$ has a service capacity of $B_i^s(t)$ requests. We denote all service rates $\{B_j^c(t)\}_{j \in \mathcal{C}}$ and $\{B_i^s(t)\}_{i \in \mathcal{S}}$ by $\mathbf{B}(t)$. Considering the resource limits on switches and controllers, we assume that $B_j^c(t) \leq b_{max}^c$ and $B_i^s(t) \leq b_{max}^s$, for some constants b_{max}^c and b_{max}^s. Besides, we also assume the existence of $E\{(A_i(t))^2\}$, $E\{(B_j^c(t))^2\}$, and $E\{(B_i^s(t))^2\}$.

2) Pre-Service Model: Besides the processing of actually arriving requests, we take a further step by considering the case when the system can predict future request arrivals in a finite number of time slots ahead.[6] Meanwhile, pre-serving future requests is also assumed applicable.[7] Particularly, each switch is installed with a learning module that actively predicts request arrivals [38], while maintaining extra buffer for predicted requests. Such requests are predicted, pre-generated (with one bit for indication in their headers), and recorded by the switch. Arriving and predicted requests, if scheduled, are appended to corresponding processing queues and later served with some queueing discipline, *e.g.*, first-in-first-out. When predicted requests arrive, if pre-served, they will be considered finished.

Formally, for each time slot t, we consider each switch i having access to its future request arrivals in a prediction window of size D_i ($< D$ for some constant D), denoted by $\{A_i(t+1), \ldots, A_i(t+D_i)\}$. In practice, one should leverage machine learning techniques such as time series forecasting methods with low computational complexities [6], and the size of the prediction window would not be very large, only few time slots. Note that the prediction may be inaccurate often times, leading to extra resource consumptions. We evaluate the impacts of mis-prediction in Section V.

With predictive scheduling, some future requests might have been pre-admitted or pre-served before time slot t, thus we use $Q_i^{(d)}(t)$ ($0 \leq d \leq D_i$) to denote the number of untreated requests in the d-th slot ahead of time t, such that

$$0 \leq Q_i^{(d)}(t) \leq A_i(t+d). \tag{1}$$

Note that $Q_i^{(0)}(t)$ denotes the number of untreated requests that actually arrive at time t. We denote the total number of

---

[6]We do not assume any particular prediction techniques in this paper, since our main focus is to explore the fundamental benefits of predictive scheduling. In practice, such prediction can be carried out by applying various machine learning techniques such as time-series prediction methods [6].

[7]The techniques of request pre-admission are still under active development. Here we take the flow management in SDN as an example. In practice, we can adopt recently developed network traffic prediction techniques [2], [22], so that switches and controllers can pre-identify long-lived or bursty flows and pre-install rules on switches to optimize the processing of such flows.

untreated requests on each switch i by $Q_i^p(t) = \sum_{d=0}^{D_i} Q_i^{(d)}(t)$. We regard $Q_i^p(t)$ as a virtual prediction queue backlog that buffers untreated future requests for switch i.

We denote the vector of all queue backlog sizes $\{Q_i^p(t)\}_{i \in \mathcal{S}}$, $\{Q_i^s(t)\}_{i \in \mathcal{S}}$, and $\{Q_j^c(t)\}_{j \in \mathcal{C}}$ by $\mathbf{Q}(t)$.

3) Scheduling Decisions: Upon new requests' arrivals, each switch should make two types of decisions.

Of the first type are **association decisions**, *i.e.*, deciding to associate with one of its potentially connected controllers or not (*devolution*). We adopt $X_{i,j}(t)$ to denote such decisions. Specifically, $X_{i,j}(t) = 1$ if switch i will be associated with controller $j \in \mathcal{C}$ in the current time slot and zero otherwise.[8] If switch i is assigned with no controllers, *i.e.*, $\sum_{j \in \mathcal{C}} X_{i,j}(t) = 0$, then it appends new requests to its local processing queue. An association is feasible if it ensures that each switch is associated with at most one controller during each time slot. The set of feasible associations is defined as follows.

$$\mathcal{A} \triangleq \{\mathbf{X} \in \{0,1\}^{|\mathcal{S}| \times |\mathcal{C}|} \mid \sum_{j \in \mathcal{C}} X_{i,j} \leq 1, \forall i \in \mathcal{S}\}. \quad (2)$$

Of the second type are **admission decisions**, *i.e.*, deciding the total number of untreated requests to be admitted, including requests that actually arrive in the current time slot and untreated future requests in its own prediction window. We use $Y_i(t)$ to denote the number of requests admitted by switch i in time slot t. Such admitted requests must include all untreated requests that have arrived, but not exceed $Q_i^p(t)$, *i.e.*,

$$Q_i^{(0)}(t) \leq Y_i(t) \leq Q_i^p(t), \quad \forall i \in \mathcal{S}, t \geq 0. \quad (3)$$

In the rest of this paper, we denote the sets $\{X_{i,j}(t)\}_{i \in \mathcal{S}, j \in \mathcal{C}}$ and $\{Y_i(t)\}_{i \in \mathcal{S}}$ by $\mathbf{X}(t)$ and $\mathbf{Y}(t)$, respectively.

4) System Workflow and Queueing Model: Basically, the system proceeds over time slots as follows. At the beginning of each time slot t, new requests arrive at switches. Scheduling decisions $[\mathbf{X}(t), \mathbf{Y}(t)]$ are then made based on instant system dynamics such as $\mathbf{Q}(t)$ and spread to all switches. Each switch i then admits $Y_i(t)$ requests, possibly including untreated future requests. All admitted future requests are marked treated and will not be served again thereafter. Meanwhile, based upon $\{X_{i,j}(t)\}_{j \in \mathcal{C}}$, each switch i sets up its association and forwards admitted requests accordingly. Next, all switches and controllers process as many requests as possible from their processing queues. At the end of time slot t, the learning module on each switch updates its new prediction of request arrivals. Accordingly, each switch's prediction window slides ahead by one time slot and new future requests are included to the corresponding prediction queues.

With the above workflow, we have the following update equations for different queue backlogs.

i) For the prediction queue with respect to switch i,

$$Q_i^p(t+1) = [Q_i^p(t) - Y_i(t)]^+ + A_i(t + D_i + 1). \quad (4)$$

ii) For the processing queue on switch $i \in \mathcal{S}$,

$$Q_i^s(t+1) = \left[Q_i^s(t) + (1 - \sum_{j \in \mathcal{C}} X_{i,j}(t))Y_i(t) - B_i^s(t) \right]^+ \quad (5)$$

[8]Our model can also be extended to handle the case of logically distributed control planes. Under such designs, each switch belongs to a distinct administrative domain. Accordingly, each switch i only has access to a subset of controllers, which can be denoted by $\mathcal{C}_i \subset \mathcal{C}$.
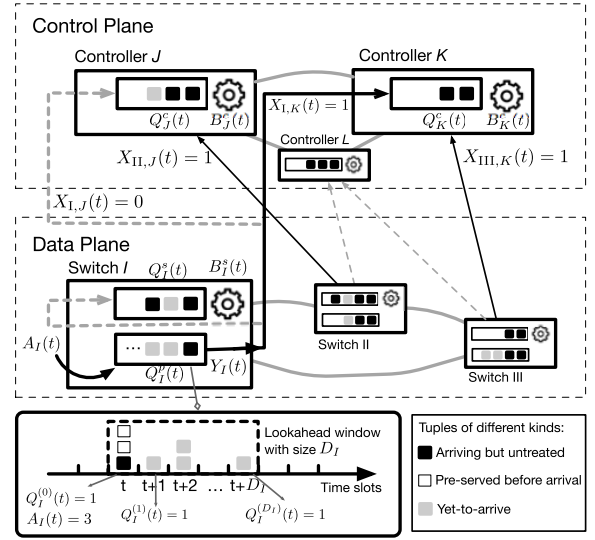

Fig. 3. An example of queueing model with predictive scheduling: At the beginning of each time slot, requests arrive at switches over time slots. Along with predicted request arrivals, they are buffered in queue $Q_i^p(t)$. Meanwhile, each switch i makes association decisions $\{X_{i,j}(t)\}_j$ and admission decision $Y_i(t)$, then either forwards requests to controllers or processes them locally. Requests are queued up and await to be processed in a first-in-first-out order.

iii) For the processing queue on controller $j \in \mathcal{C}$,

$$Q_j^c(t+1) = \left[Q_j^c(t) + \sum_{i \in \mathcal{S}} X_{i,j}(t)Y_i(t) - B_j^c(t) \right]^+. \quad (6)$$

Note that we provide an example of the above queueing model in Fig. 3.

B. Optimization Objectives

Communication Cost: Request transmissions from data plane to control plane often incur some communication costs.[9] Lower communication costs often imply shorter request response time. For each time slot t, we define $M_{i,j}(t)$ as the communication cost of forwarding one request from switch i to controller j. By denoting the set $\{M_{i,j}(t)\}_{i \in \mathcal{S}, j \in \mathcal{C}}$ by $\mathbf{M}(t)$, given decisions $\mathbf{X}(t)$ and $\mathbf{Y}(t)$, we define the total communication costs in time slot t as

$$f_p(t) \triangleq \hat{f}_p(\mathbf{X}(t), \mathbf{Y}(t)) = \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{C}} M_{i,j}(t) X_{i,j}(t) Y_i(t). \quad (7)$$

Computational Cost: Considering the scarcity of switches' computational resources, the scheduler should spare the use of switches' processing capacities. We use $P_i(t)$ to denote the computational cost for serving a request locally on switch i in time slot t. With $\mathbf{P}(t)$ as $\{P_i(t)\}_{i \in \mathcal{S}}$ and given decisions $\mathbf{X}(t)$ and $\mathbf{Y}(t)$, we define the one-time-slot computational costs as

$$g_p(t) \triangleq \hat{g}_p(\mathbf{X}(t), \mathbf{Y}(t)) = \sum_{i \in \mathcal{S}} P_i(t) \left[1 - \sum_{j \in \mathcal{C}} X_{i,j}(t) \right] Y_i(t). \quad (8)$$

Queue Stability: To ensure the timely processing of requests, it is necessary to balance queue backlogs on controllers and switches, so that no queue backlogs would suffer from being overloaded. Hence, we require the stability of all

[9]In practice, communication costs can be measured by the number of hops, round-trip times (RTT), or transmission powers, *etc.*

queue backlogs in the system. To this end, we first denote the weighted total queue backlog size in time slot t as

$$h_p(t) \triangleq \hat{h}(\mathbf{Q}(t)) = \sum_{j \in \mathcal{C}} Q_j^c(t) + \beta_1 \sum_{i \in \mathcal{S}} Q_i^s(t) + \beta_2 \sum_{i \in \mathcal{S}} Q_i^p(t), \quad (9)$$

where β_1 and β_2 are positive constants that measure the importance of balancing the backlogs of switch queues and prediction queues compared to controller queues, respectively. Then we define queue stability [29] as

$$\limsup_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{h_p(t)\} < \infty. \quad (10)$$

C. Problem Formulation

We formulate the following stochastic optimization problem that aims to minimize the long-term time-averaged expectation of the weighted sum of total communication and computation costs, while ensuring long-term queue stability.

$$\underset{\{(\mathbf{X}(t), \mathbf{Y}(t))\}_{t=0}^{T-1}}{\text{Minimize}} \quad \limsup_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \left[\mathbb{E}\{f_p(t)\} + \gamma \mathbb{E}\{g_p(t)\} \right]$$

$$\text{Subject to} \quad X_{i,j}(t) \in \mathcal{A}, \ \forall i \in \mathcal{S}, j \in \mathcal{C}, \quad (11)$$

$$Y_i(t) \in \mathbb{Z}_+, \ \forall i \in \mathcal{S}, \text{ and } (3)-(6), (10),$$

where γ is a non-negative constant that measures the scarcity of computation resources on switches.[10]

IV. ALGORITHM DESIGN AND ANALYSIS

In this section, we present detailed algorithm design to solve problem (11) with theoretical analysis and further discussion.

A. Algorithm Design

We adopt Lyapunov optimization techniques [29] to solve problem (11). First, we define the following Lyapunov function

$$L(\mathbf{Q}(t))$$
$$\triangleq \frac{1}{2} \left[\sum_{j \in \mathcal{C}} \left(Q_j^c(t) \right)^2 + \beta_1 \sum_{i \in \mathcal{S}} \left(Q_i^s(t) \right)^2 + \beta_2 \sum_{i \in \mathcal{S}} \left(Q_i^p(t) \right)^2 \right].$$
$$(12)$$

Then we define the conditional Lyapunov drift for consecutive time slots as

$$\Delta(\mathbf{Q}(t)) \triangleq E\{L(\mathbf{Q}(t+1)) - L(\mathbf{Q}(t)) \,|\, \mathbf{Q}(t)\}. \quad (13)$$

The conditional difference measures the general change in the queueing congestion state of the system. Such a difference should be as low as possible to prevent any queue backlog from being overloaded. To this end, the decision-making process should aim to minimize (13). However, this may also incur considerable communication and computational costs due to various uncertainties in the system, such as uneven request arrival traffic on different switches, inhomogeneous service capacities of controllers, and temporal variations of

communication costs. Hence, besides optimizing (13), such costs should also be considered. Given decisions $\mathbf{X}(t)$ and $\mathbf{Y}(t)$, we define the conditional drift-plus-penalty as

$$\Delta_V(\mathbf{Q}(t)) \triangleq \Delta(\mathbf{Q}(t)) + V \cdot E\{f_p(t) + g_p(t) | \mathbf{Q}(t)\}, (14)$$

where parameter V is a positive constant that controls the penalty brought by incurred system costs $f_p(t)$ (defined in (7)) and $g_p(t)$ (defined in (8)). By minimizing the upper bound of the drift-plus-penalty term (14), the time-average communication cost can be minimized while stabilizing the network of request queues [29]. We adopt the concept of *opportunistically minimizing an expectation* [29], and transform problem (11) into a series of sub-problems over time slots. Particularly, in each time slot t, we have

$$\underset{\mathbf{X}(t), \mathbf{Y}(t)}{\text{Maximize}} \sum_{i \in \mathcal{S}} l_i(t) Y_i(t) + \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{C}} u_{i,j}(t) X_{i,j}(t) Y_i(t)$$

$$\text{Subject to} \ X_{i,j}(t) \in \mathcal{A}, \ \forall i \in \mathcal{S}, j \in \mathcal{C}, \quad (15)$$

$$Y_i(t) \in \mathbb{Z}_+, \ \forall i \in \mathcal{S}, \text{ and } (3)-(6),$$

where the terms $l_i(t)$ and $u_{i,j}(t)$ are defined as

$$l_i(t) \triangleq \beta_2 Q_i^p(t) - \beta_1 Q_i^s(t) - V\gamma P_i(t), \quad (16)$$

$$u_{i,j}(t) \triangleq \left[\beta_1 Q_i^s(t) - Q_j^c(t)\right] + V\left[\gamma P_i(t) - M_{i,j}(t)\right], (17)$$

respectively. Note that parameter V is a positive parameter to determine the trade-off between queue stability and the reduction in total system costs. The objective function of (15) can be further decoupled. For each time slot t and switch i:

$$\underset{\mathbf{X}(t), \mathbf{Y}(t)}{\text{Maximize}} \ l_i(t) \cdot Y_i(t) + \left[\sum_{j \in \mathcal{C}} u_{i,j}(t) X_{i,j}(t) \right] \cdot Y_i(t)$$

$$\text{Subject to} \ X_{i,j}(t) \in \mathcal{A}, \ Y_i(t) \in \mathbb{Z}_+, \ \forall \, j \in \mathcal{C}, \quad (18)$$

$$Q_i^{(0)}(t) \leq Y_i(t) \leq Q_i^p(t).$$

Note that variables $\{X_{i,j}(t)\}_{j \in \mathcal{C}}$ and $Y_i(t)$ are coupled in the objective function, making problem (18) complicated to solve. However, by exploiting the problem structure, we show that it can be solved optimally, as follows.

Given $[\mathbf{Q}(t), \mathbf{P}(t), \mathbf{M}(t)]$, $l_i(t)$ and $u_{i,j}(t)$ are constants in time slot t. For each switch i, solving problem (18) is equivalent to finding the maximum product of two terms: 1) $Y_i(t)$ (positive and bounded), 2) $l_i(t) + \sum_{j \in \mathcal{C}} X_{i,j}(t) u_{i,j}(t)$, which depends on $\{X_{i,j}(t)\}_{j \in \mathcal{C}}$ that at most one of them equals one.

To achieve the maximum value of (18), we consider the following three cases.

Case 1: If $l_i(t) < 0$ and $l_i(t) + u_{i,j}(t) < 0$ for all j, then one should consider two sub-cases. In either case, $Y_i(t)$ should be set as its minimum $Q_i^{(0)}$, and we aim to decide $\{X_{i,j}(t)\}_{j \in \mathcal{C}}$ that maximizes the value of $l_i(t) + \sum_{j \in \mathcal{C}} X_{i,j}(t) u_{i,j}(t)$.

i) If $u_{i,j}(t) < 0$ for all j, then choose no controllers, *i.e.*, $X_{i,j}(t) = 0$ for all j.

ii) Otherwise, if $\hat{\mathcal{C}} \triangleq \{j' \in \mathcal{C} \mid u_{i,j'}(t) \geq 0\} \neq \emptyset$, then one should choose the controller $j^* \in \arg\max_{j' \in \hat{\mathcal{C}}} u_{i,j'}(t)$ by setting $X_{i,j^*}(t) = 1$ and others to be zero.[11]

---

[10]The overheads of rule installation are not considered in our system model. The reason is that in practice, such overheads are relatively small and can be further mitigated by recently proposed SDN overhead reduction techniques [31]. Moreover, our model can also be extended to handle such a case. Specifically, for each switch, we can associate each of its possible choices (including itself and its controllers) with a more general cost that takes such overheads into account, while the rest of our model requires no change.

[11]If more than one controller achieves the maximum, then the switch should choose one of them uniformly at random.

Algorithm 1 POSCAD (Predictive Online Switch-Controller Association and control Devolution) in one time slot

---

Input: Queue backlog sizes $\mathbf{Q}(t)$, computation costs $\mathbf{P}(t)$ on switches, and communication costs $\mathbf{M}(t)$ in time slot t.
Output: Admission and association decisions $\mathbf{Y}$ and $\mathbf{X}$.

1: **for** each switch $i \in \mathcal{S}$
2:    Calculate $l_i(t)$ and $u_{i,j}(t)$ for each controller $j \in \mathcal{C}$ according to (16) and (17), respectively.
3:    **if** $u_{i,j}(t) < 0$ for each controller $j \in \mathcal{C}$ **then**
4:       **if** $l_i(t) > 0$ **then**
5:          set $Y_i \leftarrow Q_i^p(t)$.
6:       **else** set $Y_i \leftarrow Q_i^{(0)}(t)$.
7:       **endif**
8:       Set $X_{i,j} \leftarrow 0, \forall j$.
9:       Admit Y_i requests from its prediction queue and
10:      append them to switch i's local queue $Q_i^s(t)$.
11:    **else** associate switch i with the controller j^* such that

$$j^* \in \arg\max_{j \in \mathcal{C}} u_{i,j}(t).$$

12:       **if** $l_i(t) + u_{i,j^*}(t) > 0$ **then**
13:          set $Y_i \leftarrow Q_i^p(t)$.
14:       **else** set $Y_i \leftarrow Q_i^{(0)}(t)$.
15:       **endif**
16:       Admit Y_i requests from its prediction queue.
17:       Forward admitted requests to controller j^* and add them to $Q_{j^*}^c(t)$; *i.e.*, set $X_{i,j^*} \leftarrow 1$ and 0 for $j \neq j^*$.
18:    **endif**
19: **endfor**
20: All switches and controllers then consume requests from their respective processing queues.

Case 2: If $l_i(t) \geq 0$ and $l_i(t) + u_{i,j}(t) < 0$ for all j, then the best choice is to choose no controllers (so that the second term equals $l_i(t)$) and set $Y_i(t)$ to its maximum $Q_i^p(t)$.

Case 3: If $\tilde{\mathcal{C}} \triangleq \{j' \in \mathcal{C} \mid l_i(t) + u_{i,j}(t) \geq 0\} \neq \emptyset$, then the optimal solution is attained by setting $X_{i,j^*}(t) = 1$ for $j^* \in \arg\max_{j' \in \tilde{\mathcal{C}}} u_{i,j'}(t)$ and others to be zero, while $Y_i(t) = Q_i^p(t)$.

In this way, problem (18) can be solved optimally. Based the above design, we propose *POSCAD*, a predictive and distributed scheme to solve problem (18). Its pseudocode is given in Algorithm 1.

B. Discussion

First, we discuss the roles of $l_i(t)$ and $u_{i,j}(t)$ in *POSCAD*, as defined in (16) and (17), respectively. On the one hand, the value of $u_{i,j}(t)$ reflects switch i's willingness of associating with controller j in time slot t. On the other hand, the value of $l_i(t)$ quantifies the weighted balance between the number of untreated requests in the prediction queue and switch i's local queue backlog size and computation cost. The larger the value of $l_i(t)$, the more untreated future requests in switch i's prediction queue, and the more requests switch i tends to admit. Note that both $l_i(t)$ and $u_{i,j}(t)$ utilize queue backlog sizes and estimate communication cost as the indicator

of congestion. Note that such information is readily available on switches and controllers at the beginning of each time slot.

Based on the above discussion, we illustrate how *POSCAD* works. If switch i decides to process new requests locally, *i.e.*, $u_{i,j}(t) < 0$ for all $j \in \mathcal{C}$, the number of admitted requests depends on the value of $l_i(t)$. If $l_i(t) > 0$, switch i would admit future requests to its local queue; otherwise, only untreated requests in current time slot will be admitted. If $u_{i,j}(t) > 0$ for some controllers, then switch i will associate with the controller j^* with the maximum $u_{i,j^*}(t)$. In this case, *POSCAD* decides request admission based on the value of $l_i(t) + u_{i,j^*}(t)$, which by (16) and (17) turns out to be

$$l_i(t) + u_{i,j^*}(t) = \beta_2 \cdot Q_i^p(t) - Q_{j^*}^c(t) - V \cdot M_{i,j^*}(t). \quad (19)$$

Intuitively, $l_i(t) + u_{i,j^*}(t)$ reflects the weighted balance among the number of all untreated requests in the prediction queue, controller j^*'s queue backlog size, and their communication cost in between. Switch i tends to admit more future requests when $l_i(t) + u_{i,j^*}(t) > 0$, *i.e.*, controller j^* possesses a queue backlog with a small size and low communication costs.

Next, we discuss the role of parameter V in *POSCAD*. Specifically, with a sufficiently large value of V, the terms related to queue backlogs in (16) and (17) become negligible. In this case, *POSCAD* inclines to neglect the workload balance among queue backlogs, and admits only arriving requests by sending them to those queues with the least costs. In contrast, if the value of V approaches zero, *POSCAD* would ignore the impact of such costs and greedily forward requests to the queue with the smallest backlog size.

C. Performance Analysis

First, we assume that the expectation of total system processing capacity is greater than that of the total request arrival rate. Then without predicted information (*i.e.*, $D_i = 0$ for each switch $i \in \mathcal{S}$), we can show that *POSCAD* achieves the classic $[O(V), O(1/V)]$ trade-off between the time-averaged expectation of total costs and total queue backlog size in the system. The proof is quite standard and thus omitted here [29].

Moreover, with predictive scheduling, *POSCAD* can achieve an even better performance by breaking the $[O(V), O(1/V)]$ backlog-cost barrier with even shorter request latencies. Intuitively, this is achieved by making each switch proactively leverage available predicted information and opportunistically exploit surplus system processing capacities to pre-serve predicted requests during each time slot. Regarding the theoretical analysis for the impacts of prediction errors, it is still an open problem. In this paper, we resort to simulation results to evaluate such impacts. More details are given in Section V.

Regarding the time complexity of *POSCAD*, from Algorithm 1, we see that *POSCAD* can be run in a distributed manner. In particular, after acquiring instant information system dynamics at the beginning of each time slot, each switch conducts decision making that is independent of each other. For each switch, the searching for the best candidate to forward requests requires only a time complexity of $O(|\mathcal{C}|)$. Such a low time complexity allows system designers to trade off only
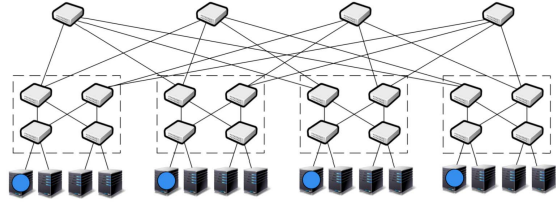
(a) Fat-tree topology with $k = 4$, where k denotes the number of switch ports. The number of core, aggregate, edge switches are $k^2/4$, $k^2/2$, and $k^2/2$, respectively. The total number of switches and hosts are $5k^2/4$ and $k^3/4$, respectively. Each edge switch is directly connected to $k/2$ hosts.
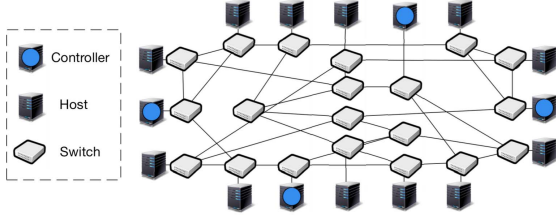
(b) Jellyfish topology with $k = 4$, where k denotes the number of switch ports. Particularly, Jellyfish topology is constructed as a random graph with $5k^2/4$ switches and $k^3/4$ hosts in total.

Fig. 4. Instances of Fat-tree and Jellyfish topologies for SDN systems. All controllers are deployed on particular hosts.

little overheads for notable improvements in system performance. More detailed discussion about the implementation of *POSCAD* is given in Section V.

V. SIMULATION RESULTS

We conduct trace-driven simulations to evaluate the performance of *POSCAD* under different settings. Particularly, in the following subsections, we first evaluate the performance of *POSCAD* without prediction (*i.e.*, with prediction window size $D_i = 0$ for each switch $i \in \mathcal{S}$). For ease of reference, we name such a special case of *POSCAD* as *GOSCAD* (Greedy Online Switch-Controller Association and control Devolution scheme). The reason is that by Algorithm 1, when $D_i = 0$ for all $i \in \mathcal{S}$, each switch i always greedily forwards requests to the controller j with the maximum positive value of $u_{i,j}(t)$ (if $u_{i,j}(t) < 0$ for all j, then process requests locally). Then we evaluate *POSCAD* under more general scenarios with perfect and imperfect predicted information, respectively.

A. Basic Settings

Topology: We evaluate the performance of *POSCAD* under four well-known data center topologies: Canonical 3-Tiered topology [4], Fat-tree [1], Jellyfish [33], and F10 [24]. We show the instances for Fat-tree and Jellyfish, respectively, in Fig. 4. To make our performance analysis fair among the four topologies, we construct them at almost the same scale, comparable to commercial data centers [4].

We deploy controllers (the control plane of the SDN system) on particular hosts, which are denoted by hosts with blue circles. Each controller has a service capacity of 600 requests per time slot (the typical setting of NOX controller [35]). Specifically, for deterministic topologies (Fat-tree, Canonical 3-Tiered, and F10), we deploy one controller in every pod;[12]

[12]In Canonical 3-Tiered topology, we regard the group of switches that belong to the same aggregation switch as one pod (including the aggregation switch itself).
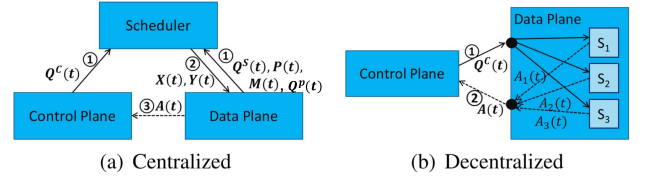

(a) Centralized   (b) Decentralized

Fig. 5. Two scheduler implementations of *POSCAD*.

for random topology (Jellyfish), we deploy the same number of controllers on hosts with non-adjacent ToRs.

Request Arrival Settings: Note that the design of *POSCAD* does not depend on particular traffic statistics. In our simulations, the request inter-arrival times follow the distribution that is drawn from measurements of real-world data centers [4]. We also consider traffic statistics that follow Poisson and Pareto distributions, since they are widely adopted in queueing network analysis. Under such scenarios, the length of each time slot is set as 10ms and the average request arrival rate on each switch is about 5.88 requests per time slot. Besides, considering the existence of hot spots (*i.e.*, some switches have intensive request arrivals) in real-world systems, we pick the first pod as the hot spot with each switch having a request arrival rate of 200 requests per time slot.

System Cost Settings: By setting parameter $\gamma = 1$, we assume the equal importance of reducing communication costs and local computational costs on switches. Given any network topology, we define the communication cost $M_{i,j}(t)$ between switch i and controller j as the length (number of hops) of shortest path from i to j. For each topology, to make switches' computational costs comparable to communication costs, we set the average unit computation cost P of switches as the average hop number between switches and controllers in the topology. In Fat-tree and F10 topologies, $P = 4.13$; in 3-Tiered and Jellyfish topologies, P equals 4.81 and 3.56, respectively.

Queueing: We treat all queue backlogs equally by setting β_1 and β_2 as 1 in (9). In practice, the values of parameter β_1 and β_2 can be tuned proportional to the ratio between the capacity of prediction queues, switches' queues, and controllers' queues. Recall that the value of parameter V determines the importance of communication cost reduction to queue stability. When applying *POSCAD*, one can tune the value of parameter V proportional to the ratio between the magnitude of queue backlog size (in number of requests) and communication cost (in milliseconds). In our simulation, the ratio is about 100 and thereby we adjust the value of parameter V from 1 to 10^4. The queueing policy is first-in-first-out (FIFO) for all queues.

Scheduler Implementations: Fig. 5 (a) and Fig. 5 (b) illustrate how *POSCAD* can be implemented in a centralized and decentralized manners, respectively. Both implementations can be applied to various systems, such as data center networks, SDN-based WANs, and SDN-based edge computing systems.

In the centralized way, the scheduler is independent of both control plane and data plane. The scheduler collects system dynamics including queue backlogs on both switches and controllers to make centralized scheduling decisions. Next, it spreads the scheduling decisions onto switches; then switches upload or locally process their requests according to
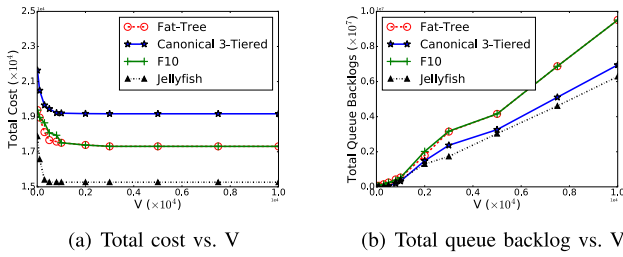
(a) Total cost vs. V   (b) Total queue backlog vs. V

Fig. 6. Performance of *GOSCAD* (*POSCAD* without prediction) under different topologies.

(a) Total cost vs. V   (b) Total queue backlog vs. V

Fig. 7. Performance of *GOSCAD* (*POSCAD* without prediction) under Fat-tree topology with Poisson and Pareto arrival processes.

such decisions. The abstract process is presented in Fig. 5 (a). The advantage of centralized architecture is that it requires no modifications on data plane, *i.e.*, all the system dynamics such as communication costs and queue backlogs can be obtained via standard OpenFlow APIs. This is well-suited for the situation where the data plane is at a large scale and switch computing resource is scarce. In fact, the scheduler could also be deployed on the control plane. There are disadvantages, too. Centralized scheduler is a potential single point of failure, or even a bottleneck with considerable computation overheads. Besides, it requires back-and-forth message exchange between the SDN system and the scheduler, thus leading to longer latencies. Specifically, under such an implementation, it takes extra bandwidth consumptions and latencies to obtain instant system dynamics (system costs and backlog sizes) from the control plane and the data plane. Nonetheless, the mechanism for probing instant system dynamics can be implemented in a scalable fashion. In particular, by deploying *POSCAD* over a group of dedicated servers, each server only needs to probe instant dynamics from part of switches and controllers. Based on probed information, each server can conduct independent decision making for switches.

In the decentralized way, as Fig. 5 (b) shows, switches periodically update their information about queue backlogs from control plane; meanwhile, each of them can make scheduling decisions independently. Though this way requires modification on switches, it still enjoys the following advantages. It requires less amounts of message exchange than that in the centralized way, thus switches would response even faster to network events. On the other hand, the computation of decision-making in both schemes is distributed across switches, leading to better scalability and fault tolerance. In our simulations, we choose the decentralized implementation for *POSCAD*.

B. Evaluation of GOSCAD (POSCAD Without Prediction)

Fig. 6 (a) and Fig. 6 (b) show how different values of parameter V affect the long-term time-averaged total costs of communication and computation, and the total queue backlog size in the system. In general, as the value of parameter V grows from 1 to 10^4, we see a rapid reduction in total system costs and a linear increase in the total queue backlog size. However, as the value of V continues to increase, the reduction in total system costs diminishes, while the queue backlog size keeps going up. Note that this verifies our previous theoretic results on the $[O(V), O(1/V)]$ trade-off between the
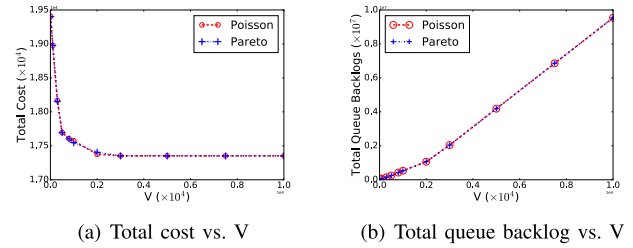
time-averaged expectation of total costs and total queue backlog size in the system. We discuss them in detail as follows.

First, recall that the value of parameter V actually controls the switches' willingness of uploading requests. For switches that are close to controllers (their communication costs are less than the average), large values of V make them incline to uploading requests, unless all controllers become heavily loaded. For switches that are distant from controllers, local processing is a better choice. Consequently, large values of V generally lead to lower communication costs. However, as the value of V becomes sufficiently large, switches either greedily process requests locally or upload them to the nearest controllers, leading to hot spots on particular controllers with ever-increasing backlog sizes. In this case, according to Little's theorem [21], request response time increases as well.

Fig. 6 (a) presents how the long-term time-averaged total costs of communication and local computation (on switches) with different values of V in the four topologies. We make the following observations.

First, as the value of V varies from 1 to 10^4, it shows that the total cost goes down gradually. This is consistent with our previous theoretic analysis. The intuition behind such decline is as follows. Remind that V controls the switches' willingness of uploading requests. For switches that are close to controllers (their communication cost is less than the average), large V makes them unwilling to process requests locally unless the controllers are heavily loaded. As the value of V increases, those switches will choose to upload requests to further reduce the costs since for those switches, communication costs are less than the computation costs.

Second, the total cost in 3-Tiered topology is greater than the other schemes', because it has a higher unit computational cost ($P = 4.81$ compared to 4.13 and 3.56 for Fat-tree/F10 and Jellyfish). Meanwhile, switches in 3-Tiered topology usually take longer paths to controllers and incur more communication costs compared to other topologies.

Third, the total costs under Jellyfish topology are significantly lower (up to 21%) than the others, as Jellyfish generally has a shorter average path between switches and controllers than the other deterministic topologies of the same scale [33].

Fig. 7 (a) and Fig. 7 (b) show the performance of *GOSCAD* under Fat-tree topology with other two request arrival processes. The results show that *GOSCAD* (*i.e.*, *POSCAD* without prediction) requires no prior knowledge about the statistics of request arrival dynamics and still achieves the $[O(V), O(1/V)]$ backlog-cost trade-off.

(a) Canonical 3-Tiered topology   (b) Fat-tree topology
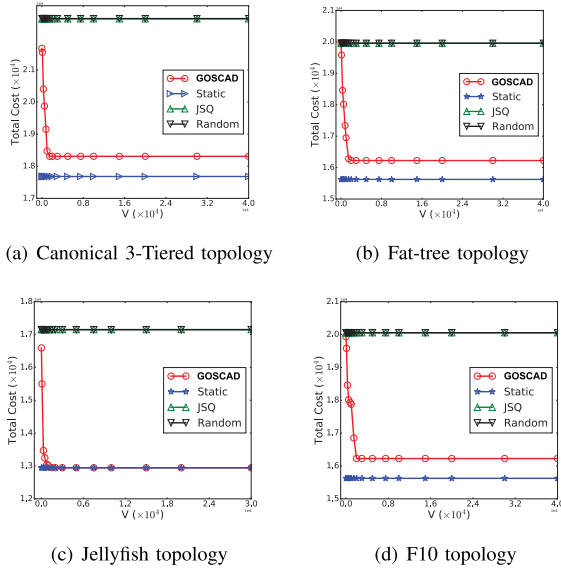
(c) Jellyfish topology   (d) F10 topology

Fig. 8. Comparison of communication costs among four scheduling schemes under different topologies. Note that *GOSCAD* corresponds to the special case of *POSCAD* without prediction.

(a) Canonical 3-Tiered topology   (b) Fat-tree topology

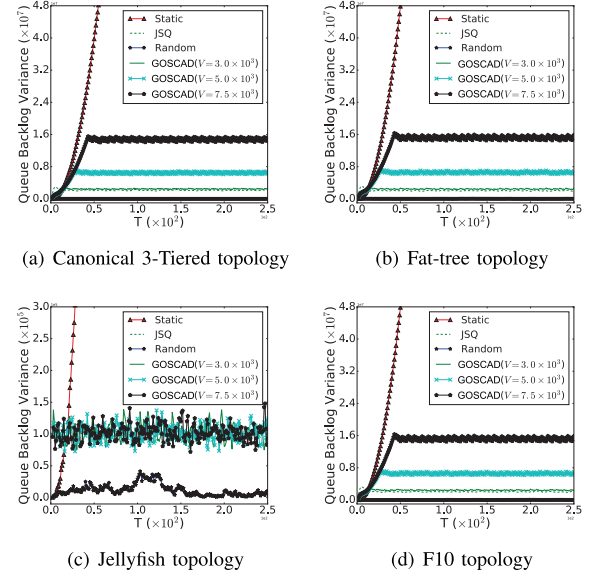(c) Jellyfish topology   (d) F10 topology

Fig. 9. Variances of queue backlog size comparison incurred by four scheduling schemes under different topologies. Note that *GOSCAD* corresponds to the special case of *POSCAD* without prediction.

C. Performance Comparison Against Baseline Schemes

In Fig. 8-Fig. 11, we compare the performance of *GOSCAD* (*POSCAD* without prediction) against three baseline schemes: *Static*, *Random*, and *JSQ* (Join-the-Shortest-Queue). Particularly, in *Static* scheme, each switch i chooses the controller j with the minimum communication costs. In *Random* scheme, each switch is scheduled to pick up a controller uniformly at random during each time slot. In *JSQ* scheme, each switch picks the controller with the shortest queue backlog. Note that the baseline schemes do not conduct control devolution. To make the comparison fair, we set the unit computational cost $P = 10^{28}$ for all switches. This makes the costs of local computation prohibitively high; as a result, under *GOSCAD*, switches would keep uploading new requests to associated controllers. That being said, *GOSCAD* degenerates into a dynamic switch-controller association scheme. Note that such settings also emulate the scenarios in which control devolution is not supported.

From Fig. 8, we see that *Static* achieves the minimum total costs, since it greedily associates switches to controllers with minimum communication costs. On the other hand, both *Random* and *JSQ* incur much higher communication costs. The reason is that their decisions make no use of system dynamics about communication costs. Compared to baseline schemes, *GOSCAD* cuts down the communication costs as the value of V increases. Eventually, its induced communication costs stays 5.7% above the curve of *Static*. In fact, such a gap is the price taken to maintain queue stability. Note that when the value of V increases, *GOSCAD*'s behavior becomes increasingly similar to *Static*, which focuses more on the reduction in total system costs. The difference emerges when some controllers' queue backlog sizes exceed some threshold (about twice the value of V in our simulations). In such cases, *Static* would continue pursuing the minimum communication costs and send requests to controllers which are heavily loaded but with the

lowest communication costs. Consequently, such controllers will become overloaded and even lead to system breakdown. Under *GOSCAD*, however, some switches would rather send requests to controllers with higher costs but a smaller queue backlog size, to ensure queue stability. Such illustration is supported by the results from Fig. 9. From the results, we see that the larger the value of parameter V, the greater the variance of queue backlog sizes among different controllers. In the extreme case (*Static*), the variance grows remarkably. Unlike *Static*, *GOSCAD* constantly ensures queue stability. Besides, we also notice that such a gap is much smaller under Jellyfish topology, because therein the communication costs are more even to each switch, resulting in more balanced queue backlogs in the system.

Insight: As the value of V increases, *GOSCAD* achieves a tunable trade-off between system cost reduction and queue stability. The gap between *GOSCAD* and the minimum costs is the price which has to be taken to maintain queue stability.

Fig. 10 shows the incurred total queue backlog sizes under GOSCAD and the baseline schemes. We see that under different topologies, both *Random* and *JSQ* maintain the total queue backlog sizes at a low level, which is consistent with the result in Fig. 9. In contrast, *Static* induces significantly higher queue backlog sizes, due to its greediness to send requests to controllers with lowest costs, leading to imbalanced queue backlogs. Intuitively, the more balanced the queue backlogs across controllers, the more controller resources are utilized, and hence a smaller total queue backlog size.

As for *GOSCAD*, under deterministic topologies (3-Tiered, Fat-tree, and F10), we observe a reduction in the backlog size at the very beginning, then a linear increase after reaching a valley at around 10^3. The explanation is as follows. When the value of V is small, switches prefer controllers with shorter queues.[13] Recall that a switch's scheduling decision is

---

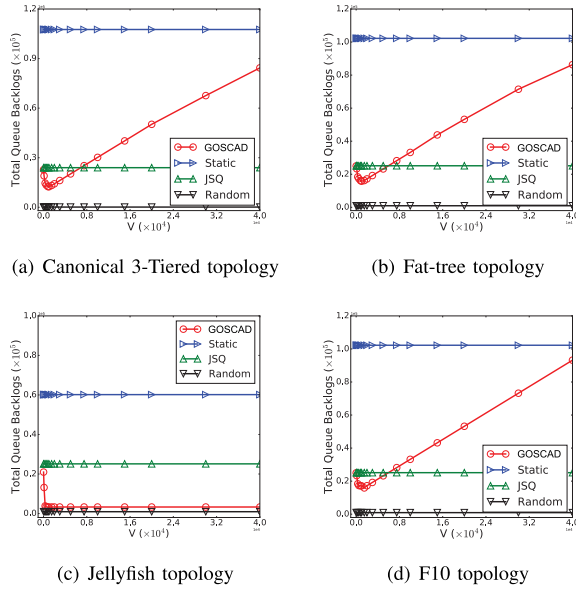[13]Note that *JSQ* is a special case of *GOSCAD* with $V = 0$.

Fig. 10. Total queue backlog size comparison among four scheduling schemes under different topologies. Note that *GOSCAD* corresponds to the special case of *POSCAD* without prediction.
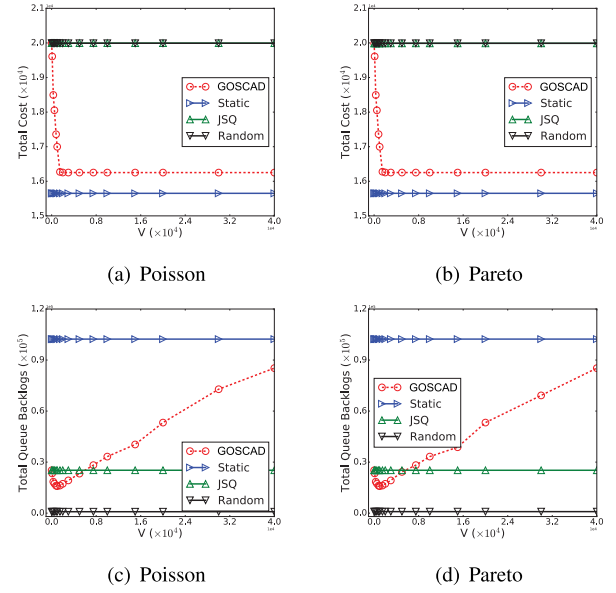

Fig. 11. Performance comparison among different schemes under Fat-tree topology, with Poisson and Pareto request arrivals, respectively. Note that *GOSCAD* corresponds to the special case of *POSCAD* without prediction.

independent of the others'. This will lead to requests being intensively forwarded to few controllers. Then controllers close to hot spots are more likely to be heavily loaded. As the value of V becomes larger, some switches would reach a tipping point and choose other controllers instead. As a result, this would make controllers' backlogs more balanced, and hence smaller total queue backlog sizes. When the value of V continues to grow, switches turn to forward requests with the aim to minimize communication costs. As a result, requests will constantly accumulate on some particular controllers, leading to queue backlog imbalance across controllers and hence the growth in the total queue backlog size.

When *GOSCAD* is applied in Jellyfish, its curve is very different from other three topologies. We can see the significant reduction at the beginning and then it stays at a low level thereafter. We find that for Fat-tree, F10, and Canonical 3-Tiered, they have higher variance in the number of hops between switches and controllers; as a result, with the existence of hot spots and the aim to minimize communication costs, more requests accumulate on some particular controllers, inducing increased queue backlog sizes. For Jellyfish, however, because incoming requests are spread more evenly among controllers, increasing the value of V has no significant impacts on the skewness of controller queue backlogs.

Insight: In practice, one can tune the value of parameter V to be proportional to the ratio between the magnitude of queue backlog size and communication cost, to achieve a significant reduction in the total system costs while maintaining balanced queue backlogs among controllers.

In addition, we also conduct simulations under two kinds of request arrival processes, *i.e.*, *Poisson* and *Pareto* processes, which are widely adopted in traffic analysis. We only show the simulation results under Fat-tree topology, because the simulation results in the other three topologies are qualitatively similar. From Fig. 11, we find that all schemes perform qualitatively consistent under different arrival processes.

D. Evaluation of POSCAD

1) Prediction Settings: We set switches' prediction window sizes by sampling them uniformly at random in $[0, 2 \times D]$, with mean D. The values of D range from 0 to 20. Besides, considering that prediction errors are inevitable in practice, we evaluate POSCAD's performance with prediction errors at different levels. We use e_t to denote the prediction deviation for time slot t, *i.e.*, the difference between the number of predicted and actual arrivals. Particularly, $e_t = 0$ indicates that the prediction is perfect. When the prediction is imperfect, $e_t > 0$ implies that the future request arrivals are over-estimated; otherwise, they are under-estimated. We also assume e_t to be *i.i.d.* over time slots, and generated by 1) sampling a value x_t from some probability distribution with zero mean and then 2) obtaining e_t by rounding x_t, *i.e.*, $e_t \triangleq \text{Round}(x_t)$. The second step is due to that the probability distribution may be continuous but the number of requests must be an integer. Then we define prediction error rate r, *i.e.*, the probability of mis-prediction, as $r \triangleq \Pr\{e_t \neq 0\} = \Pr\{x_t \notin (-0.5, 0.5)\}$. In our simulation, we sample x_t from a *normal distribution* with zero mean and variance σ^2. Accordingly, we have

$$r = 2[1 - \Phi(0.5/\sigma)], \qquad (20)$$

where $\Phi(\cdot)$ is the CDF of *standard normal distribution*. Based on (20), the error rate r can be adjusted by choosing an appropriate value of σ. We pick error rates from 0% to 50%.

2) Response Time Metric: As for evaluation metrics, note that the total queue backlog sizes in the non-prediction case ($D = 0$) and the predictive case ($D > 0$) are incomparable. The reason is that with predictive scheduling, the total queue backlog size, as defined in (9), also includes future untreated requests. All such requests are still counted, although they have not actually arrived or even not been pre-served. The comparison is even more inappropriate with predictive scheduling in the presence of errors, where some future requests may
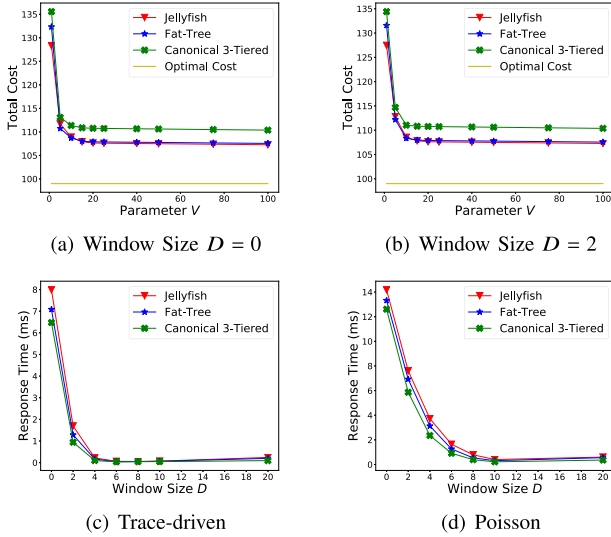
(a) Window Size $D = 0$

(b) Window Size $D = 2$

(c) Trace-driven

(d) Poisson

Fig. 12. Performance under perfect prediction settings.

Fig. 13. Average response time vs. prediction error rate under Fat-tree topology with different parameter V and prediction window size D.

not even exist due to over-estimation. Hence, instead of total queue backlog size, we evaluate *POSCAD* in terms of average request response time, since *POSCAD* is promising to reduce request response times by exploiting future information and pre-serving future requests using idle system resources. In our simulations, we define a request's response time as the number of time slots from its actual arrival to its eventual completion. If pre-served before its actual arrival, a request will be responded instantly and hence experiencing a near-zero response time. In our simulations, the average request response time is obtained over completed requests.

3) Evaluation With Perfect Prediction: To explore the benefits of predictive scheduling, we first consider the case where switches have perfect information about future request arrivals in the lookahead window, *i.e.*, with an error rate of $r = 0$. Considering the similarity between curves under Fat-Tree and F10, we omit the results for F10 in the following.

Time-averaged total costs vs. V: We compare the total costs incurred by *POSCAD* with different values of V under different topologies in Fig. 12. Particularly, we compare the results with prediction window sizes $D = 0$ (non-prediction) and $D = 2$ in Fig. 12 (a) and Fig. 12 (b), respectively. We see a rapid reduction in total system costs when the value of V increases from 1 to 20. As the value of V continues to increase, the reduction eventually diminishes with 10% gap to the optimal total costs (denoted by the solid horizontal line). Comparing Fig. 12 (a) and Fig. 12 (b), we see that the incurred total costs are almost the same even when the window size D is increased. With such observations, we have the following insight.

Insight: As the value of parameter V increases, *POSCAD* reduces the time-averaged total costs and achieves the optimal system costs asymptotically under different topologies. Note that the gap between the converged costs and the optimal costs is the price paid for stabilizing queue backlogs in the system. In the meantime, compared to the non-prediction case, *POSCAD* performs predictive scheduling without increasing the total costs in the system.
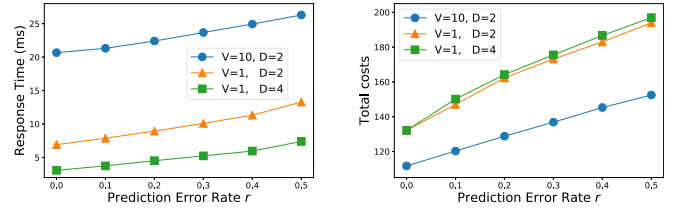
Average response time vs. prediction window size: Next, we focus on the impact of different prediction window sizes on the average request response time. Fig. 12 (c) and Fig. 12 (d) present the results from simulations with trace-driven and Poisson arrival process, each with an average arrival rate of 5.88 requests per time slot, respectively. Fig. 12 (c) shows the curves under trace-driven settings. We see a sharp decrease of the average request response time from around 14ms to less than 1ms, as the window size rises from 0 (non-prediction) to 6 under different topologies. As the window size continues growing, the reduction stops and the average response time converges (0.45ms). The results are similar in Fig. 12 (d).

Insight: Fig. 12 shows the benefits of predictive scheduling. By exploiting predictive information to pre-serve requests, *POSCAD* effectively reduces the average response time without incurring extra system costs. Moreover, with a mild-value of lookahead window size, *POSCAD* achieves a significant reduction in the average request response time.

4) Evaluation With Imperfect Prediction: Prediction errors are inevitable in practice. Hence, we are also interested in the robustness of POSCAD against such errors. In the following, we vary error rate from 0% to 50%.

Average response time vs. prediction error rate: Fig. 13 shows the system performance with prediction error rate r ranging from 0% (perfect prediction) to 50% under *POSCAD* with various settings. On the one hand, with parameter $V = 10$ and window size $D = 2$, we observe a growth (6.21ms) of the average response time as the prediction error rate rises from 0% to 50%. This suggests that prediction errors lead to an increased request response time. With prediction error rate r fixed as 10% and a prediction window size $D = 2$, we see that the response time (21.26ms) under $V = 10$ is higher than that (7.9ms) under $V = 1$. The result suggests that one can reduce request response time by decreasing the value of parameter V. The reason is that a smaller value of V leads to a smaller total queue backlog size which, by Little's theorem [21], implies a lower average response time. Besides, by fixing the prediction error rate (take $r = 10\%$ as an example), we see that the larger the window size, the shorter the response time. This implies that more future information conduces to response time reduction by up to 50% even under inaccurate prediction. On the other hand, we see that higher prediction error rates lead to higher system costs, since the system needs to allocate more resources to process mis-predicted requests. Besides, there are no significant costs ($\sim 2\%$) incurred as the value of window size D increases. Meanwhile, increasing the value of V results in a notable reduction of response time. Such results show that the advantage of predictive scheduling

mainly lies in response time reduction. Its impact is minor on system costs because of the trade-off between cost reduction and queue stability.

Insight: *POSCAD* is robust against prediction errors. By choosing a smaller value of V and enlarging the prediction window size, *POSCAD* can effectively eliminate the negative effect of prediction errors and shortens request response time.

VI. Related Work

Regarding *switch-controller association*, the usual design choice is to make a static switch-controller association [17], [34]. However, solutions with static association are often inflexible when dealing with temporal variations of request traffic, thereby inducing workload imbalance across controllers and an increased request processing latency. To mitigate such issues, Dixit *et al.* proposed an elastic distributed controller architecture with an efficient protocol design for switch migration among controllers [10]. However, the design for *switch-controller association* still remained unresolved. Later, Krishnamurthy *et al.* took a further step by formulating the controller association problem as an integer linear problem with a prohibitively high computational complexity [19]. A local search algorithm was proposed to find the best possible association within a given time limit (*e.g.*, 30 seconds). Wang *et al.* modeled the controller as an $M/M/1$ queueing system [36]. By formulating the association problem with a steady-state objective function as a many-to-one stable matching problem with transfers, they developed a novel two-phase algorithm that connects stable matching to utility-based game theoretic solutions, *i.e.*, coalition formation game with Nash stable solutions. Later, they extended the problem with an aim to minimize the long-term costs in SDN systems [37]. By decomposing it into a series of per-time-slot controller assignment sub-problems, Wang *et al.* applied receding horizon control techniques to solve the problem. In parallel, Filali *et al.* [11] formulated the problem as an one-to-many matching game, then they developed a matching-based algorithm that achieves load-balancing by assigning a minimum quota of workload to each controller. Lyu *et al.* [25] presented an adaptive decentralized approach for joint switch-controller association and controller activation with periodic on-off control to save operational costs. Regarding control devolution, most works have focused on static delegation of certain network functions to switches [9], [12], [44]. Some recent works [40] have proposed more flexible devolution schemes based on the distribution of network states or workloads on controllers in real time.

Compared to existing works, the focus of our work is on the joint design of dynamic switch-controller association and control devolution with performance analysis, while investigating the benefits of predictive scheduling in SDN systems. Particularly, without prediction, our schemes are not dependent on any particular network traffic distributions. However, when adopted in real scenarios with dramatic traffic variations [23], [41], network traffic prediction would inevitably lead to prediction errors. Nonetheless, our proposed scheme is still robust against such prediction inaccuracy, as illustrated in Section V.

VII. Conclusion

In this paper, we studied the problem of dynamic switch-controller association and control devolution, and investigated the benefits of predictive scheduling for SDN systems. We proposed *POSCAD*, an efficient online joint control scheme that solves the problem through a series of online decision making in a distributed manner. Our theoretical analysis showed that without prediction, *POSCAD* achieves near-optimal total costs with queue stability guarantee. Furthermore, with predictive scheduling, *POSCAD* achieves an even better performance with a significant reduction in request response time. We conducted extensive simulations to verify the effectiveness of *POSCAD*. Our results showed that with mild-value of future information, *POSCAD* incurs a significant reduction in request response time, even in face of mis-prediction. Notably, the focus of this work is the exploration of the *fundamental* limits of the benefits of predictive scheduling in SDN systems. Therefore, we did not consider testbed-based experimental verifications in this work. Nonetheless, it can be an interesting direction for future work.

References

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

[2] A. Azzouni and G. Pujolle, "NeuTM: A neural network-based framework for traffic matrix prediction in SDN," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, Apr. 2018, pp. 1–5.

[3] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN control: Survey, taxonomy, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 333–354, 1st Quart., 2018.

[4] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th Annu. Conf. Internet Meas. (IMC)*, 2010, pp. 267–280.

[5] P. Berde *et al.*, "ONOS: Towards an open, distributed SDN OS," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 1–6.

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

[7] J. Broughton. (2016). *Netflix Adds Download Functionality*. [Online]. Available: https://technology.ihs.com/586280/netflix-adds-download-support

[8] N. Chen, J. Comden, Z. Liu, A. Gandhi, and A. Wierman, "Using predictions in online optimization: Looking forward with an eye on the past," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 44, no. 1, pp. 193–206, 2016.

[9] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *Proc. ACM SIGCOMM*, 2011, pp. 254–265.

[10] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2013, pp. 7–12.

[11] A. Filali, A. Kobbane, M. Elmachkour, and S. Cherkaoui, "SDN controller assignment and load balancing with minimum quota of processing capacity," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.

[12] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proc. ACM HotSDN*, 2012, pp. 19–24.

[13] L. Huang, S. Zhang, M. Chen, and X. Liu, "When backpressure meets predictive scheduling," *IEEE/ACM Trans. Netw.*, vol. 24, no. 4, pp. 2237–2250, Aug. 2016.

[14] X. Huang, S. Bian, Z. Shao, and H. Xu, "Dynamic switch-controller association and control devolution for SDN systems," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2017, pp. 1–6.

[15] X. Huang, S. Bian, Z. Shao, and H. Xu, "Predictive switch-controller association and control devolution for SDN systems," in *Proc. Int. Symp. Qual. Service*, Jun. 2019, pp. 1–10.

[16] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, 2013.

[17] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *Proc. USENIX OSDI*, 2010, pp. 1–14.

[18] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[19] 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.

[20] 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.

[21] J. D. Little, "A proof for the queuing formula," *Oper. Res.*, vol. 9, no. 3, pp. 383–387, 1961.

[22] C. Liu, A. Malboubi, and C.-N. Chuah, "OpenMeasure: Adaptive flow measurement & inference with online learning in SDN," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2016, pp. 47–52.

[23] 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.

[24] V. Liu, D. Halperin, A. Krishnamurthy, and T. E. Anderson, "F10: A fault-tolerant engineered network," in *Proc. USENIX NSDI*, 2013, pp. 399–412.

[25] X. Lyu, C. Ren, W. Ni, H. Tian, R. P. Liu, and Y. J. Guo, "Multitimescale decentralized online orchestration of software-defined networks," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 12, pp. 2716–2730, Dec. 2018.

[26] A. Masood and A. Hashmi, "AIOps: Predictive analytics & machine learning in operations," in *Cognitive Computing Recipes*. Springer, 2019, pp. 359–382.

[27] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[28] S. Nanda, F. Zafari, C. DeCusatis, E. Wedaa, and B. Yang, "Predicting network attack patterns in SDN using machine learning approach," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2016, pp. 167–172.

[29] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synth. Lectures Commun. Netw.*, vol. 3, no. 1, pp. 1–211, Jan. 2010.

[30] K. Phemius, M. Bouet, and J. Leguay, "DISCO: Distributed multi-domain SDN controllers," in *Proc. IEEE Netw. Operations Manage. Symp. (NOMS)*, May 2014, pp. 1–4.

[31] L. Sanabria-Russo, J. Alonso-Zarate, and C. Verikoukis, "SDN-based pro-active flow installation mechanism for delay reduction in IoT," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–6.

[32] M. A. S. Santos, B. A. A. Nunes, K. Obraczka, T. Turletti, B. T. de Oliveira, and C. B. Margi, "Decentralizing SDN's control plane," in *Proc. 39th Annu. IEEE Conf. Local Comput. Netw.*, Sep. 2014, pp. 402–405.

[33] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *Proc. USENIX NSDI*, 2012, pp. 225–238.

[34] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proc. Internet Netw. Manage. Conf. Res. Enterprise Netw.*, 2010, pp. 1–6.

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

[36] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic SDN controller assignment in data center networks: Stable matching with transfers," in *Proc. IEEE 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.

[37] T. Wang, F. Liu, and H. Xu, "An efficient online algorithm for dynamic SDN controller assignment in data center networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2788–2801, Oct. 2017.

[38] J. Xie *et al.*, "A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 393–430, 1st Quart., 2019.

[39] Y. Xu, V. Mahendran, and S. Radhakrishnan, "SDN docker: Enabling application auto-docking/undocking in edge switch," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2016, pp. 864–869.

[40] J. Yang, X. Yang, Z. Zhou, X. Wu, T. Benson, and C. Hu, "FOCUS: Function offloading from a controller to utilize switch power," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2016, pp. 199–205.

[41] X. Yi, F. Liu, J. Liu, and H. Jin, "Building a network highway for big data: Architecture and challenges," *IEEE Netw.*, vol. 28, no. 4, pp. 5–13, Jul. 2014.

[42] H. Yu, M. H. Cheung, L. Huang, and J. Huang, "Power-delay tradeoff with predictive scheduling in integrated cellular and Wi-Fi networks," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 4, pp. 735–742, Apr. 2016.

[43] S. Zhang, L. Huang, M. Chen, and X. Liu, "Proactive serving decreases user delay exponentially: The light-tailed service time case," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 708–723, Apr. 2017.

[44] K. Zheng, L. Wang, B. Yang, Y. Sun, Y. Zhang, and S. Uhlig, "Lazy Ctrl: Scalable network control for cloud data centers," in *Proc. IEEE 35th Int. Conf. Distrib. Comput. Syst.*, Jun./Jul. 2015, pp. 788–789.
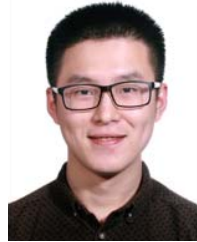
Xi Huang (Member, IEEE) received the B.Eng. degree from the Software Institute, Nanjing University, China, in 2014. He is currently pursuing the Ph.D. degree with the School of Information Science and Technology, ShanghaiTech University. He was also a Visiting Student with the RISELab (previously AMPLab), EECS Department, UC Berkeley, in 2017. His research interests include datacenter networks, software-defined networking, data stream processing systems, and edge intelligence.

Simeng Bian (Student Member, IEEE) received the B.Eng. degree from the School of Software and Microelectronics, Northwestern Polytechnical University, China, in 2016. She is currently pursuing the Ph.D. degree with the School of Information Science and Technology, ShanghaiTech University. Her current research interests include network function virtualization, fog computing, and reinforcement learning.

Ziyu Shao (Senior Member, IEEE) received the B.S. and M.Eng. degrees from Peking University in 2001 and 2004, respectively, and the Ph.D. degree from The Chinese University of Hong Kong in 2010. Then, he worked as a Post-Doctoral Researcher with The Chinese University of Hong Kong from 2011 to 2013. Meanwhile, he was a Visiting Post-Doctoral Researcher with the Department of EE, Princeton University, in 2012. He was also a Visiting Professor with the EECS Department, UC Berkeley, in 2017. He is currently an Assistant Professor with the School of Information Science and Technology, ShanghaiTech University, Shanghai, China. His current research interest includes learning and optimization for intelligent networks, including cloud computing and fog/edge computing.

Hong Xu (Senior Member, IEEE) received the B.Eng. degree from The Chinese University of Hong Kong in 2007, and the M.A.Sc. and Ph.D. degrees from the University of Toronto in 2009 and 2013, respectively. He is currently a Faculty Member of the City University of Hong Kong. He will join the Department of Computer Science and Engineering, The Chinese University of Hong Kong, as an Associate Professor, after September 2020. His research interest includes computer networking and systems, particularly data center networks and big data systems. He is a member of the ACM. He was a recipient of the Early Career Scheme Grant from the Hong Kong Research Grants Council in 2014. He received several best paper awards, including the IEEE ICNP 2015 Best Paper Award.