Predictive Switch-Controller Association and Control Devolution for SDN Systems

Xi Huang[1], Simeng Bian[1], Ziyu Shao[1], Hong Xu[2]
[1]School of Information Science and Technology, ShanghaiTech University
[2] NetX Lab, Department of Computer Science, City University of Hong Kong
Email: {huangxi,biansm,shaozy}@shanghaitech.edu.cn, henry.xu@cityu.edu.hk

***Abstract*—For software-defined networking (SDN) systems, to enhance the scalability and reliability of control plane, existing solutions mainly 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, leading to 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 control scheme that conducts dynamic switch-controller association and dynamic control devolution. In addition, the fundamental benefits of predictive scheduling to SDN systems still remains unexplored. In this paper, we identify the non-trivial trade-off in such joint design and formulate it as a stochastic optimization problem which aims to minimize time-average total system costs and ensure long-term queueing stability. By exploiting unique problem structure, we propose a greedy online switch-controller association and control devolution (GOSCAD) scheme, which solves the problem through a series of online distributed decision making. Theoretical analysis shows that GOSCAD can achieve near-optimal total system costs with tunable trade-off for queueing stability. Extending GOSCAD with predictive scheduling, we propose predictive online switch-controller association and control devolution (POSCAD), which proactively pre-serves requests to further enhance system performance. Results from extensive simulations verify the effectiveness of both proposed schemes. Notably, with mild-value of future information, POSCAD induces significant reduction in request response time, even with mis-prediction.**

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

I. Introduction

In 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 control plane from data plane [18]. In such a way, control plane maintains a centralized view of the whole network and processes requests that are constantly generated from SDN-enabled switches in the data plane; meanwhile, data plane only needs to carry out basic network functions such as monitoring and packet forwarding.

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

To address such issues, existing solutions basically fall into two categories. One is to implement the control plane as a distributed system with multiple controllers [12] [23]; then each switch associates with more than one controller for fault-tolerance and load balancing [14] [7] [13] [25] [26] [9] [17]. The other is to devolve some request processing that does not require global information from controllers to switches, to reduce workloads on control plane [6] [10] [30].

Regarding switch-controller association, *i.e.*, the first category of solutions, the usual design choice is to make a static switch-controller association [12] [23]. However, solutions with static association are often inflexible when dealing with temporal variations of request traffic, thereby inducing workload imbalance across controllers and 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 [7]. 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 prohibitively high computational complexity [13]. 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 [25]; they formulated the association problem with a steady-state objective function as a many-to-one stable matching problem with transfers. Then 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 [26]. 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.* [9] formulated the problem as an one-to-many matching game, then developed another matching-based algorithm that achieves load-balancing by assigning minimum quota of workload to each controller. Lyu *et al.* [17] 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, the second category of solutions, most works have focused on static delegation of certain network functions to switches [6], [10], [30]. Some recent works [27] have even proposed more flexible schemes to perform delegation based on real-time distribution of network states or workloads on controllers.

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

- ⋄ Instead of conducting deterministic switch-controller association with infrequent re-association [13] [25], can we directly perform dynamic association with respect to request traffic variation? 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 uncertainties in request traffic is one of the key factors that bring challenges to SDN system design, then if they can be learned, what are the fundamental benefits of such predicted information to SDN systems?

Notably, the last question is motivated by the recent growing interests in applying predictive scheduling [8] [4] based on machine learning techniques to improve system performance. For instance, to promote quality-of-experience, Netflix predicts user behavior and preferences, then preloads videos onto user devices. Despite the wide adoption of such prediction-based approaches [28] [29] [5] [21] [19] in recent years, the fundamental benefits of predictive scheduling for SDN systems still remains unexplored.

In this paper, we focus on general SDN systems with requests dynamically generated from switches in the data plane for 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 time 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 limited time slots ahead through lightweight prediction modules with recently developed machine learning techniques [3]. Further, we assume such future requests can be generated and pre-served before their arrival, and, if mis-predicted, they will induce extra system costs of communication and computation. [2] Under such settings, we open up a new perspective to answer those questions. The following summarize our contributions.

- ⋄ **Modeling and Formulation:** We formulate the problem stated above as a stochastic optimization problem that aims to minimize time-average total system costs with long-term queueing stability constraint. Through a careful choice in the granularity of modeling and decision making, *i.e.*, to characterize system dynamics at request level and scheduling on a time-slot basis, we achieve a decent balance between modeling accuracy and decision making complexity.
- ⋄ **Algorithm Design:** By adopting existing techniques [20] and exploiting unique sub-problem structure, we propose a greedy online switch-controller association and control devolution (*GOSCAD*) scheme, which makes association and devolution decisions in a distributed manner with limited information about instant system dynamics.
- ⋄ **Performance Analysis:** We conduct theoretical analysis, which shows that *GOSCAD* yields a tunable trade-off between $O(1/V)$ deviation from minimum long-term average sum of communication costs and computational costs and $O(V)$ bound for long-term average queue backlog size. We also provide detailed discussion about the insights and implementation issues in practice.
- ⋄ **New Degree of Freedom in the Design Space of SDN Systems**: By adopting the idea of sliding lookahead window model [11], we further propose *POSCAD*, a predictive scheme that goes beyond *GOSCAD* by extending it with predictive scheduling. *POSCAD* effectively reduces request response time by proactively pre-serving requests with surplus system resources. To our best knowledge, this is 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.
- ⋄ **Experimental Evaluation and Verification:** We conduct extensive simulations to verify the effectiveness of *GOSCAD* and *POSCAD*. Our results show that under various settings, both schemes achieve near-optimal system costs while maintaining a tunable trade-off with queueing stability. Furthermore, given only mild-value of predicted information, *POSCAD* induces a significant reduction in request response time, even in face of mis-prediction.

We organize the rest of paper as follows. Section II illustrates the basic idea, modeling, and problem formulation of dynamic switch-controller association and control devolution. Then Section III shows the algorithm design of *GOSCAD* and its performance analysis. Section IV extends previous formulation with predictive scheduling, followed by the algorithm design of *POSCAD*. Section V presents simulation results, while Section VI concludes this paper.

II. Problem Formulation

In this section, we first present a motivating example to reveal the non-trivial trade-off in the joint design of dynamic switch-controller association and control devolution; then we introduce our system model and problem formulation.

A. Motivating Example

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

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

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

[2] The scenario that some future requests may not be pre-served due to their statefulness is also a special case of our model.

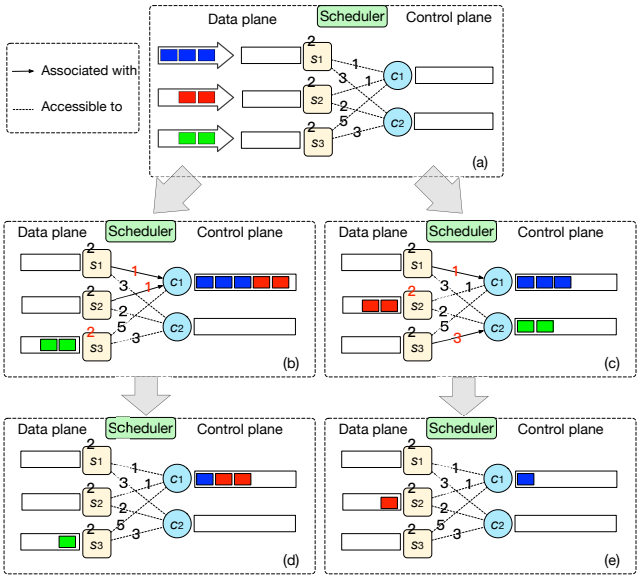


Fig. 1. *An SDN system with request-level switch-controller association and control devolution within one time slot*. There are three switches(s_1, s_2, s_3), two controllers(c_1, c_2), and one global scheduler. Switches and controllers maintain queue backlogs to buffer requests. Between each pair of 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.

Next, we switch to the consequences of the scheduling decisions to the whole system. In Figure 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 Figure 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 Figure 1 (c), the decision (denoted by X_2) includes two associations, (s_1, c_1) and (s_3, c_2) (s_2 processes requests locally). Figure 1(e) shows that X_2 does better than X_1 with only two finished requests at the end of the time slot. However, this is achieved at higher total costs of 13.

The above two observations show that there is a non-trivial trade-off between the system costs reduction and minimizing total queue backlog size. In the next section, we develop a fine-grained queueing model to characterize such a trade-off, with key notations summarized in Table I.

B. System Model

We consider an SDN system that evolves over time slots, indexed by $t \in \{0, 1, 2, \dots\}$. The system's control plane comprises a set C of physically distributed controllers, while the data plane consists of a group of SDN-enabled switches $\mathcal{S}$. Each switch i keeps a local processing queue of backlog size $Q_i^s(t)$, while each controller j maintains a queue backlog $Q_j^c(t)$ that buffers requests from data plane. We denote $\{Q_j^c(t)\}_j$ by $\mathbf{Q}^c(t)$, $\{Q_i^s(t)\}_i$ by $\mathbf{Q}^s(t)$, and $[\mathbf{Q}^s(t), \mathbf{Q}^c(t)]$ by $\mathbf{Q}(t)$.

For each time slot t, there is a number of $A_i(t)$ ($\leq a_{max}$ for some constant a_{max}) new requests arriving at each switch i. Such arrivals $A_i(t)$ are assumed i.i.d. over time slots. We denote $\mathbf{A}(t) \triangleq \{A_i(t)\}_i$. Besides, all the requests are assumed homogeneous and can be handled by both switches and controllers, though our model can be directly extended to scenarios with inhomogeneous 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 has a service capacity of $B_i^s(t)$ requests. We denote all such service rates by $\mathbf{B}(t) \triangleq \{B_j^c(t)\}_{j\in C} \cup \{B_i^s(t)\}_{i\in\mathcal{S}}$. Considering the resource limit 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. In addition, we also assume the existence of $E\{(A_i(t))^2\}$, $E\{(B_j^c(t))^2\}$, and $E\{(B_i^s(t))^2\}$.

TABLE I
KEY NOTATIONS

Symbol	Description
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
$A_i(t)$	The number of request arrivals on switch i in time t
$B_i^s(t)$	Service capacity of switch i in time slot t
$B_j^c(t)$	Service capacity of controller j in time slot t
$M_{i,j}(t)$	Per-request communication cost between switch i and controller j
$P_i(t)$	Per-request computational cost on switch i
$X_{i,j}(t)$	Association decision for switch i and controller j in time slot t
$Y_i(t)$	Admission decision for switch i

The overall system proceeds as follows. At the beginning of time slot t, the system scheduler collects system dynamics $(\mathbf{A}(t), \mathbf{B}(t), \mathbf{Q}(t))$ and makes scheduling decisions, denoted by an association matrix $\mathbf{X}(t) \in \{0, 1\}^{|\mathcal{S}|\times|C|}$. In particular, $X_{i,j}(t) = 1$ if switch i will be associated with controller j during current time slot and zero otherwise. 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 characterized as follows.

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

The detailed implementation of the scheduling procedure will be discussed in Section V. Based on such decisions, switches forward requests accordingly. For switch i, it sends new requests to controller j if $X_{i,j} = 1$. If switch i is assigned with no controllers, *i.e.*, $\sum_{j\in C} X_{i,j} = 0$, then it appends new requests to its local processing queue. Meanwhile, switches and controllers serve as many requests from their respective processing queues as possible. Therefore, the update equations for queue backlogs on switch i and controller j are given by

$$Q_i^s(t+1) = \left[Q_i^s(t) + \left(1 - \sum_{j\in C} X_{i,j}(t)\right)A_i(t) - B_i^s(t)\right]^+, \quad (2)$$

$$Q_j^c(t+1) = \left[Q_j^c(t) + \sum_{i\in\mathcal{S}} X_{i,j}(t)\cdot A_i(t) - B_j^c(t)\right]^+, \quad (3)$$

respectively, where the operator $[\,x\,]^+ \triangleq \max(x, 0)$.

C. Optimization Objectives

Communication Cost: Request transmissions from data plane to control plane often induce some communication cost.[3]

[3]In practice, communication costs may refer to the number of hops, round-trip times (RTT), or transmission powers, *etc.*

Lower communication costs often imply shorter response time to requests and less energy consumption. 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. Denoting the set $\{M_{i,j}(t)\}_{i\in\mathcal{S},j\in C}$ by $\mathbf{M}(t)$, Fixing some association $\mathbf{X}\in\mathcal{A}$, we define the total communication costs in time slot t as

$$f_{\mathbf{X}}(t)=\hat{f}(\mathbf{X},\mathbf{A}(t))\triangleq\sum_{j\in C}\sum_{i\in\mathcal{S}}M_{i,j}(t)\cdot X_{i,j}\cdot A_i(t), \tag{4}$$

where we can regard $M_{i,j}(t)$ as the price of transmitting one request from switch i to controller j.

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

$$g_{\mathbf{X}}(t)=\hat{g}(\mathbf{X},\mathbf{A}(t))\triangleq\sum_{i\in\mathcal{S}}P_i(t)\cdot\Big(1-\sum_{j\in C}X_{i,j}\Big)\cdot A_i(t). \tag{5}$$

Queueing 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 queue backlogs in the system. We define queueing stability [20] as

$$\limsup_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\left[\sum_{i\in\mathcal{S}}E\left\{Q_i^s(t)\right\}+\sum_{j\in C}E\left\{Q_j^c(t)\right\}\right]<\infty. \tag{6}$$

D. Problem Formulation

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

$$\begin{aligned}\underset{\{\mathbf{X}(t)\in\mathcal{A}\}_t}{\text{Minimize}}\quad&\limsup_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\left[\mathbb{E}\left\{f_{\mathbf{X}(t)}(t)\right\}+\gamma\mathbb{E}\left\{g_{\mathbf{X}(t)}(t)\right\}\right]\\ \text{subject to}\quad&(2),(3),(6),\end{aligned} \tag{7}$$

where γ is a non-negative constant that weighs the scarcity of computation resources on switches.

III. Algorithm Design and Analysis

In this section, we present detailed algorithm design to solve problem (7) with theoretical analysis and further discussion.

A. Algorithm Design

We adopt Lyapunov optimization techniques [20] to solve problem (7). First, we define the following Lyapunov function

$$L(\mathbf{Q}(t))\triangleq\frac{1}{2}\left[\sum_{j\in C}\left(Q_j^c(t)\right)^2+\sum_{i\in\mathcal{S}}\left(Q_i^s(t)\right)^2\right]. \tag{8}$$

Then we define the conditional Lyapunov drift given $\mathbf{Q}(t)$ for two consecutive time slots as

$$\Delta\left(\mathbf{Q}(t)\right)\triangleq E\Big\{L(\mathbf{Q}(t+1))-L(\mathbf{Q}(t))\,|\,\mathbf{Q}(t)\Big\}. \tag{9}$$

Such a conditional difference measures the general change in queues' congestion state. In general, such difference should be as low as possible to prevent queues $\mathbf{Q}^s(t)$ and $\mathbf{Q}^c(t)$ from being overloaded. To this end, the decision making process should be conducted to minimize (9); however, this may incur considerable communication costs and computational costs as well, because of various uncertainties in the system, such as the uneven request arrival traffic on different switches, inhomogeneous service capacities on controllers, and the time-varying communication costs. Hence, besides queueing stability, the decision making should also factor in the increase of such system costs. Given any decisions $\mathbf{X}\in\mathcal{A}$, we define the one-time-slot conditional drift-plus-penalty function as

$$\Delta_V(\mathbf{Q}(t))\triangleq\Delta(\mathbf{Q}(t))+V\cdot E\left\{f_{\mathbf{X}}(t)+g_{\mathbf{X}}(t)|\mathbf{Q}(t)\right\}. \tag{10}$$

where parameter V is a positive constant that weighs the penalty brought by communication costs between switches and controllers ($f_{\mathbf{X}}(t)$) and the local computation costs on switches due to control devolution ($g_{\mathbf{X}}(t)$). By minimizing the upper bound of the drift-plus-penalty term (34), the time-average communication costs can be minimized while stabilizing the network of request queues [20]. We adopt the concept of *opportunistically minimizing an expectation* [20], and transform the long-term stochastic optimization problem (7) into a series of drift-plus-penalty minimization problem over time slots. We refer detailed transformation procedure to Appendix A. Particularly, in each time slot t,

$$\begin{aligned}\underset{\mathbf{X}\in\mathcal{A}}{\text{Minimize}}\quad&V\cdot\Big(\hat{f}(\mathbf{X},\mathbf{A}(t))+\hat{g}(\mathbf{X},\mathbf{A}(t))\Big)+\\&\sum_{j\in C}Q_j^c(t)\cdot\Big[\sum_{i\in\mathcal{S}}X_{i,j}\cdot A_i(t)\Big]+\\&\sum_{i\in\mathcal{S}}Q_i^s(t)\cdot\Big[(1-\sum_{j\in C}X_{i,j})\cdot A_i(t)\Big].\end{aligned} \tag{11}$$

By rearranging the terms, we can rewrite problem (11) as

$$\begin{aligned}\underset{\mathbf{X}\in\mathcal{A}}{\text{Minimize}}\ \sum_{i\in\mathcal{S}}\Big[V\cdot P_i(t)+Q_i^s(t)\Big]A_i(t)+\sum_{i\in\mathcal{S}}\sum_{j\in C}\Big[VM_{i,j}(t)\\+Q_j^c(t)-VP_i(t)-Q_i^s(t)\Big]X_{i,j}A_i(t).\end{aligned} \tag{12}$$

Notice that the first term $\sum_{i\in\mathcal{S}}[VP_i(t)+Q_i^s(t)]$ has nothing to do with decision variables $\mathbf{X}$. Thus, we only focus on minimizing its second term. To simplify notations, we define

$$\begin{aligned}\omega(i,j)&\triangleq-\Big(V\cdot M_{i,j}(t)+Q_j^c(t)-V\cdot P_i(t)-Q_i^s(t)\Big)\\&=V\cdot\big(P_i(t)-M_{i,j}(t)\big)+\Big(Q_i^s(t)-Q_j^c(t)\Big).\end{aligned} \tag{13}$$

Equivalently, we rewrite problem (35) as

$$\underset{\mathbf{X}\in\mathcal{A}}{\text{Maximize}}\quad\sum_{i\in\mathcal{S}}\sum_{j\in C}\omega(i,j)X_{i,j}A_i(t). \tag{14}$$

The above problem can further decoupled into subproblems, each of which, *i.e.*, maximizing $\sum_{j\in C}\omega(i,j)X_{i,j}A_i(t)$, can be solved by individual switches. For each $i\in\mathcal{S}$, to maximize the value of $\sum_{j\in C}\omega(i,j)X_{i,j}A_i(t)$, we split C into two disjoint sets $\mathcal{J}_1^i$ and $\mathcal{J}_2^i$, *i.e.* $\mathcal{J}_1^i\bigcup\mathcal{J}_2^i=C$, such that

$$\mathcal{J}_1^i\triangleq\{j\in C\,|\,\omega(i,j)<0\}\text{ and }\mathcal{J}_2^i\triangleq\{j\in C\,|\,\omega(i,j)\geq 0\}. \tag{15}$$

Then we have

$$\sum_{j\in C}\omega(i,j)\mathbf{X}_{i,j}A_i(t)=\left\{\sum_{j\in\mathcal{J}_1^i}\omega(i,j)X_{i,j}+\sum_{j\in\mathcal{J}_2^i}\omega(i,j)X_{i,j}\right\}A_i(t). \tag{16}$$

Next, we show how to maximize (16) with $\mathbf{X} \in \mathcal{A}$. We use $\mathbf{X}^*$ to denote maximizer of (16) and consider two cases.

i) If $\mathcal{J}_2^i = \emptyset$, *i.e.*, $\omega(i,j) < 0$ for all $j \in C$, then the only way to maximize (16) is setting $X_{i,j}^* = 0$ for all $j \in C$.

ii) If $\mathcal{J}_2^i \neq \emptyset$, then $X_{i,j}$ should be handled for $j \in \mathcal{J}_1^i$ and $j \in \mathcal{J}_2^i$ separately.

a) For $j \in \mathcal{J}_1^i$, to maximize (16), it is not hard to see one should set $X_{i,j}^* = 0$ for all $j \in \mathcal{J}_1^i$.

b) For $j \in \mathcal{J}_2^i$, $\omega(i,j) \geq 0$. Set $X_{i,j^*}^* = 1$ for such j^* that[4]

$$j^* \in \arg\max_{j \in \mathcal{J}_2^i} \quad \omega(i,j), \tag{17}$$

and $X_{i,j}^* = 0$ for $j \in \mathcal{J}_2^i - \{j^*\}$.

Thus, we acquire $\mathbf{X}^*$, the maximizer to (16), or equivalently, the minimizer to (35). We propose a Greedy Online Switch-Controller Association and control Devolution (*GOSCAD*) scheme, with its pseudocode shown in Algorithm 1.

Algorithm 1 GOSCAD (Greedy Online Switch-controller Association and control Devolution)

Input: At the beginning of each time slot t, system dynamics, *e.g.*, *i.e.* $\mathbf{Q}^c(t)$, $\mathbf{Q}^s(t)$, $\mathbf{A}(t)$, $\mathbf{P}(t)$, and $\mathbf{M}(t)$ are collected.
Output: A scheduling association $\mathbf{X} \in \{0,1\}^{|\mathcal{S}|\times|C|}$.

1: Initially, set $X_{i,j} \leftarrow 0$ for all $(i,j) \in \mathcal{S} \times C$
2: **for** each switch $i \in \mathcal{S}$ **do**
3: Divide all controllers C into two sets $\mathcal{J}_1^i$ and $\mathcal{J}_2^i$, where $\mathcal{J}_1^i \leftarrow \{j \in C \mid \omega(i,j) < 0\}$ and $\mathcal{J}_2^i \leftarrow \{j \in C \mid \omega(i,j) \geq 0\}$.
4: If $\mathcal{J}_2^i = \emptyset$, then skip the current iteration.
5: If $\mathcal{J}_2^i \neq \emptyset$, then choose a controller $j^* \in \mathcal{J}_2^i$ such that

$$j^* \in \arg\max_{j \in C} \omega(i,j),$$

6: and set $X_{i,j^*} \leftarrow 1$.
7: **end for**
8: **return** $\mathbf{X}$
Switches forward new requests according to decisions $\mathbf{X}$. *Meanwhile, controllers and switches serve requests from their processing queues.*

Discussion: For switch i and controller j, $\omega(i,j)$ reflects the willingness of switch i to send new requests to controller j. By (13), such willingness is mainly determined by two factors.

i) One is the relative size of switch i's local computation costs $P_i(t)$ compared to its communication costs $M_{i,j}(t)$ to controller j. If in the current time slot, local processing is more costly than uploading requests to j, then the uploading the new requests is surely more favorable.

ii) Befores, switch i also needs to factor in the backlog size $Q_i^s(t)$ of its local processing queue and that $(Q_j^c(t))$ of controller j. If switch i's queue backlog is more loaded than controller j, *i.e.*, $Q_i^s(t) \geq Q_j^c(t)$, then it will incline to uploading requests to mitigate its workload pressure.

Next, we consider such willingness under different scenarios.

i) If switch i has a considerable local computation costs and larger queue backlog size than controller j, then its willingness with respect to j, $\omega(i,j)$, will be positive.

ii) If switch i has surplus computation resources (smaller computation cost) and smaller backlog size than controller j, it will tend to process the new requests locally.

iii) However, if switch i has surplus computation resources but its backlog size is greater than controller j, then its willingness depends on which factor weighs more: to reduce system costs or to keep its local backlog small. Such relative importance is determined by parameter V. The greater the value of V, the more focus will be put on reducing system costs and hence local processing. The smaller the value of V, the switch will upload requests for queueing stability concern.

iv) If the local computation is costly but switch i has a smaller backlog size than controller j, then switch i's willingness also depends on the value of parameter V.

Basically, *GOSCAD* is online since all decision makings are based on the instant system dynamics within the current time slot. Besides, *GOSCAD* is greedy. For switch i, it determines the switch's association and devolution by the following process. Among switch i's potential controllers, if there's no such controller that switch i is willing to upload its requests, *i.e.*, $\omega(i,j) < 0$ for all j, then switch i will process requests locally. Otherwise, *GOSCAD* greedily associates switch i to the controller with the highest willingness.

B. Performance Analysis

By assuming that the expectation of total system processing capacity is greater than that of the total request arrival rate, we show that *GOSCAD* achieves the classic $[O(V), O(1/V)]$ trade-off between the time-average expectation of total costs and total queue backlog size in the system. We refer the proof to Appendix B.

Regarding the time complexity of *GOSCAD*, from Algorithm 1, we see that GOSCAD can be run in a distributed manner. In particular, after acquiring the instant information system dynamics at the beginning of each time slot, each switch conducts decision making independent of each other. For each switch, the computation and searching for the best candidate to process requests require only time complexity in $O(|C|)$. Such a low time complexity allows system designers to trade off only little overheads for notable improvement in system performance. We present further discussion about its implementation in Section V.

IV. Extension with Predictive Scheduling

In this section, we take a further step by integrating predictive scheduling with the joint control of switch-controller association and control devolution. We provide a motivating example to illustrate the potential benefits of predictive scheduling in Appendix C.

A. Extended Model with Predictive Scheduling

1) Pre-service Model: Besides the processing of actually arriving requests, as defined in Section II-B, 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.[5]

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

[5] 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 [3].

Meanwhile, pre-serving future requests is also assumed applicable. In particular, each switch is embedded with a learning module that predicts the request arrival, and maintains extra queue backlog to buffer yet-to-arrive requests. Such requests are predicted, pre-generated (marked as predicted by using only one bit in their headers), and recorded by the switch. Arriving and predicted requests, if scheduled, are appended to corresponding processing queues and later served by some queueing policy, *e.g.*, first-in-first-out (FIFO). 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 $\{A_i(t+1), \ldots, A_i(t+D_i)\}$. There are two things to note in practice. First, due to the resource scarcity on switches and the uncertainty in request arrivals, one should leverage time series forecasting techniques with low computational complexity [3], and the size of the prediction window would not be very large, only few time slots. Second, the prediction may be inaccurate often times, which can lead to extra resource consumption. We evaluate the impact of mis-prediction in Section V.

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

$$0 \le Q_i^{(d)}(t) \le A_i(t+d). \tag{18}$$

Note that $Q_i^{(0)}(t)$ denotes the number of untreated requests that actually arrive in time slot t. We denote the total number of 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 backlogs $\{Q_i^p(t)\}_{i\in\mathcal{S}}$, $\{Q_i^s(t)\}_{i\in\mathcal{S}}$, and $\{Q_j^c(t)\}_{j\in C}$ by $\mathbf{Q}(t)$.

2) Scheduling Decisions: Upon new requests' arrivals, each switch should make two decisions with predictive scheduling.

The first is **admission decision**, *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 to be admitted from switch i in time slot t. These admitted requests must include all untreated requests that actually arrive, but not exceed $Q_i^p(t)$, *i.e.*,

$$Q_i^{(0)}(t) \le Y_i(t) \le Q_i^p(t), \quad \forall i \in \mathcal{S}, t \ge 0. \tag{19}$$

The second is **association decisions**, *i.e.*, deciding to process these admitted requests locally (control devolution), or associate with and forward requests to one of its potentially connected controllers. We adopt $X_{i,j}(t)$ to denote such decisions, as defined in Section II-B.

We denote sets $\{Y_i(t)\}_{i\in\mathcal{S}}$ and $\{X_{i,j}(t)\}_{i\in\mathcal{S},j\in C}$ by $\mathbf{Y}(t)$ and $\mathbf{X}(t)$, respectively.

3) 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 notified to all the switches. Each switch $i \in \mathcal{S}$ 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.

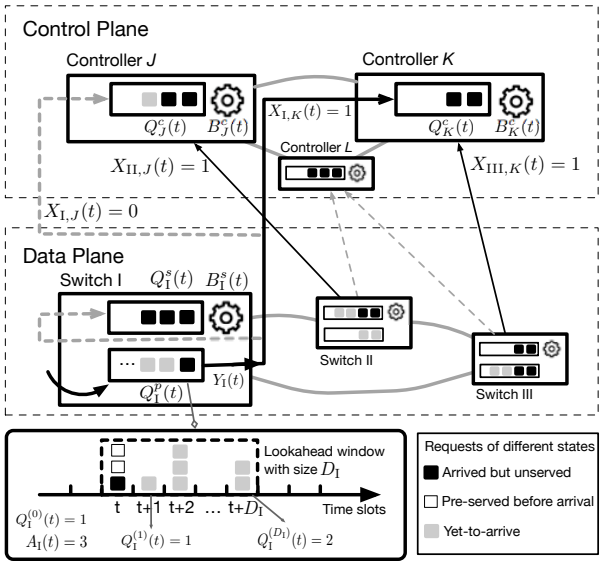


Fig. 2. An example of queueing model with predictive scheduling: At the beginning of each time slot, requests arrive at switches over time slots. Along with predicted request arrivals, they are buffered in queue $Q_i^p(t)$. Meanwhile, each switch i makes the admission decision $X_i(t)$ and association decision $Y_{i,*}(t)$, then either forwards requests to controllers or process them locally. Requests are queued up and await to be processed in a first-in-first-out order.

Meanwhile, based upon $\{X_{i,j}(t)\}_{j\in C}$, each switch i sets up its association and forward the admitted requests accordingly. Next, all switches and controllers serve requests from their own processing queues. At the end of time t, the learning module on each switch updates its new prediction about the request arrival. Accordingly, switches' prediction windows move one slot ahead and the new future requests are appended 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,

$$Q_i^p(t+1) = \left[Q_i^p(t) - Y_i(t)\right]^+ + A_i(t+D_i+1). \tag{20}$$

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

$$Q_i^s(t+1) = \left[Q_i^s(t) + (1 - \sum_{j\in C} X_{i,j}(t))Y_i(t) - B_i^s(t)\right]^+. \tag{21}$$

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

$$Q_j^c(t+1) = \left[Q_j^c(t) + \textstyle\sum_{i\in\mathcal{S}} X_{i,j}(t)Y_i(t) - B_j^c(t)\right]^+. \tag{22}$$

Note that previous queueing model in Section II-B is a special case of the above model: in each time slot, the switch only admits requests arriving in current time slot. Figure 2 shows an example of the above predictive queueing model.

4) Optimization Objectives:

Communication Cost: Recall that $M_{i,j}(t)$ denotes the communication costs of sending a request from switch i to controller j. With predictive scheduling, we define the total communication costs in time t as

$$f_p(t) = \hat{f}_p\left(\mathbf{X}(t), \mathbf{Y}(t)\right) \triangleq \sum_{i\in\mathcal{S}}\sum_{j\in C} M_{i,j}(t)X_{i,j}(t)Y_i(t). \tag{23}$$

Computational Cost: Recall that $P_i(t)$ denotes the computational cost for keeping a request locally on switch i. Then we define the total computational costs as

$$g_p(t) = \hat{g}_p\left(\mathbf{X}(t), \mathbf{Y}(t)\right) \triangleq \sum_{i\in\mathcal{S}} P_i(t)\Big[1 - \sum_{j\in C} X_{i,j}(t)\Big] Y_i(t). \tag{24}$$

Queueing Stability: We first denote the weighted total queue backlog size in time slot t as

$$\begin{aligned} h_p(t) &= \hat{h}(\mathbf{Q}(t)) \\ &\triangleq \sum_{j\in C} Q_j^c(t) + \beta_1 \sum_{i\in\mathcal{S}} Q_i^s(t) + \beta_2 \sum_{i\in\mathcal{S}} Q_i^p(t). \end{aligned} \tag{25}$$

where β_1 and β_2 are positive constants that weigh the importance of balancing the queue backlogs on switches compared to controllers'. We define queueing stability [20] as

$$\limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\left\{h_p(t)\right\} < \infty. \tag{26}$$

B. Problem Formulation

We extend the problem formulation in (7) and write

$$\begin{aligned} \underset{\{(\mathbf{X}(t),\mathbf{Y}(t))\}_{t=0}^{T-1}}{\text{Minimize}} \quad & \limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \left[\mathbb{E}\left\{f_p(t)\right\} + \gamma \mathbb{E}\left\{g_p(t)\right\}\right] \\ \text{Subject to} \quad & X_{i,j}(t) \in \{0,1\}, \ \ \forall i \in \mathcal{S}, j \in C, \\ & Y_i(t) \in \mathbb{Z}_+, \ \ \forall i \in \mathcal{S}, \text{ and } (19)-(22), (26), \end{aligned} \tag{27}$$

where γ is a non-negative constant that weighs the scarcity of computation resources on switches.

C. Algorithm Design

Following similar steps in Section III-A, we transform the long-term stochastic optimization problem (27) into a series of subproblems over time slots; *i.e.*, for each time slot t,

$$\begin{aligned} \underset{\mathbf{X}(t),\mathbf{Y}(t)}{\text{Maximize}} \quad & \sum_{i\in\mathcal{S}} l_i(t) Y_i(t) + \sum_{i\in\mathcal{S}} \sum_{j\in C} u_{i,j}(t) X_{i,j}(t) Y_i(t) \\ \text{Subject to } & X_{i,j}(t) \in \{0,1\}, \ \ \forall i \in \mathcal{S}, j \in C, \\ & Y_i(t) \in \mathbb{Z}_+, \ \ \forall i \in \mathcal{S}, \text{ and } (19)-(22), \end{aligned} \tag{28}$$

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

$$l_i(t) \triangleq \beta_2 Q_i^p(t) - \beta_1 Q_i^s(t) - V\gamma P_i(t), \tag{29}$$

$$u_{i,j}(t) \triangleq \left[\beta_1 Q_i^s(t) - Q_j^c(t)\right] + V\left[\gamma P_i(t) - M_{i,j}(t)\right], \tag{30}$$

respectively. As to *GOSCAD*, parameter V is a positive constant to determine the trade-off between queueing stability and reduction in total system costs. The objective function of (28) can be further decoupled. For each time slot t and switch i,

$$\begin{aligned} \underset{\mathbf{X}(t),\mathbf{Y}(t)}{\text{Maximize}} \quad & l_i(t)\cdot Y_i(t) + \Big[\sum_{j\in C} u_{i,j}(t) X_{i,j}(t)\Big] \cdot Y_i(t) \\ \text{Subject to } & X_{i,j}(t) \in \{0,1\}, \ \ Y_i(t) \in \mathbb{Z}_+, \ \ \forall\ j \in C, \\ & \sum_{j\in C} X_{i,j}(t) \le 1, \ \ Q_i^{(0)}(t) \le Y_i(t) \le Q_i^p(t). \end{aligned} \tag{31}$$

Notice that variables $X_{i,*}(t)$ and $Y_i(t)$ are coupled in the objective function, making problem (31) even more complicated. However, by exploiting the subproblem's structure, it can still 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 t. For switch i, problem (31) is equivalent to finding the maximum product of two terms: 1) $Y_i(t)$ (positive and bounded), 2) $l_i(t) + \sum_{j\in C} X_{i,j}(t) u_{i,j}(t)$, dependent on $X_{i,*}(t)$ that at most one of them equals one.

To achieve the maximum, we consider three cases.

i) 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 $X_{i,*}(t)$ that maximizes $l_i(t) + \sum_{j\in C} X_{i,j}(t) u_{i,j}(t)$.
   a) If $l_i(t) > \max_j(l_i(t) + u_{i,j}(t))$, *i.e.*, if $u_{i,j}(t) < 0$ for all j, then choose no controllers, *i.e.*, $X_{i,j}(t) = 0$ for all j.
   b) Else, if $\hat{C} \triangleq \{j' \in C \mid u_{i,j'}(t) \ge 0\} \ne \emptyset$, then one should choose the controller $j^* \in \arg\max_{j'\in\hat{C}} u_{i,j'}(t)$ by setting $X_{i,j^*}(t) = 1$ and others to be zero.

ii) If $l_i(t) \ge 0$ 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 $Q_i^p(t)$.

iii) If $\tilde{C} \triangleq \{j' \in C \mid l_i(t) + u_{i,j}(t) \ge 0\} \ne \emptyset$, then the optimal solution is attained by setting $X_{i,j^*}(t) = 1$ for $j^* \in \arg\max_{j'\in\tilde{C}} u_{i,j'}(t)$ and others to be zero. Still, $Y_i(t)$ should be set as $Q_i^p(t)$.

In this way, problem (31) can be solved optimally. We propose a Predictive Online Switch-controller Association and Control Devolution (*POSCAD*) scheme and we show its pseudocode in Algorithm 2.

D. Discussion

First, we note that, without predictive scheduling, *POSCAD* degenerates to *GOSCAD* by setting $Y_i(t)$ as $A_i(t)$ for all time slot t; *i.e.*, *POSCAD* only admits and treats requests that actually arrive and in the current time slot. However, when integrated with predictive scheduling, besides association, *POSCAD* also decides when to pre-serve future requests. In such a way, *POSCAD* goes beyond *GOSCAD* by breaking the $[O(V), O(1/V)]$ backlog-cost trade-off with even lower request response time. This is achieved by pre-admitting future requests opportunistically and exploiting the surplus system processing capacity in every time slot to pre-serve them. We verify the effectiveness of *POSCAD* in Section V.

Next, we discuss the roles of $l_i(t)$ and $u_{i,j}(t)$ in *POSCAD*, as defined in (29) and (30). On one hand, similar to *GOSCAD*, $u_{i,j}(t)$ reflects switch i's willingness of associating with controller j in time t. On the other hand, $l_i(t)$ 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 costs as the indicator of congestion, which are directly attainable from switches and controllers at the beginning of each time slot. However, for information such as service capacity, they remain unknown until the end of time slot, after requests being processed.

Algorithm 2 POSCAD (Predictive Online Switch-Controller Association and control Devolution) in each time slot t

Input: Current queue backlog sizes $\mathbf{Q}(t)$, computation costs $\mathbf{P}(t)$ on switches, and communication costs $\mathbf{M}(t)$.
Output: Admission and association decisions.

for each switch $i \in \mathcal{S}$
  Calculate $l_i(t)$ and $u_{i,j}(t)$ for every controller $j \in C$ according to (29) and (30), respectively.
  if $u_{i,j}(t) < 0$ for every controller $j \in C$ **then**
    if $l_i(t) > 0$ **then**
      admit all $Q_i^p(t)$ requests from i's prediction queue.
    else admit $Q_i^{(0)}(t)$ untreated requests in time slot t.
  endif
    Append all admitted requests to the local processing queue $Q_i^s(t)$ on switch i.
  else associate switch i with the controller j^* such that

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

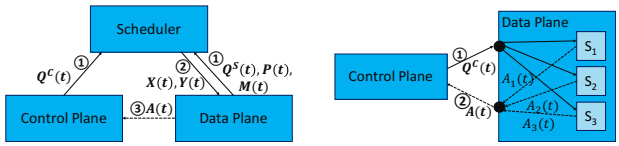
    if $l_i(t) + u_{i,j^*}(t) > 0$ **then**
      admit all $Q_i^p(t)$ requests from i's prediction queue.
    else admit $Q_i^{(0)}(t)$ untreated requests in time slot t.
  endif
    Forward all admitted requests to controller j^* and append them to $Q_{j^*}^c(t)$ therein.
  endif
endfor
All switches and controllers then consume requests from their respective processing queues.

The above illustration leads to a better understanding of how *POSCAD* works. If switch i decides to process new requests locally, *i.e.*, $u_{i,j}(t) < 0$ for all $j \in C$, then the number of admitted requests depends on $l_i(t)$. 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 $l_i(t) + u_{i,j^*}(t)$, which according to (29) and (30), turns out to be

$$l_i(t) + u_{i,j^*}(t) = \beta_2 Q_i^p(t) - Q_{j^*}^c(t) - V \cdot M_{i,j^*}(t). \tag{32}$$

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 costs 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 a low communication cost.

Similar to *GOSCAD*, the value of *POSCAD*'s parameter V determines the relative importance of minimizing communication costs to maintaining queueing stability for switches and controllers. With an extremely large value of V, the terms related to queue backlogs in (29) and (30) become negligible. In this case, *POSCAD* inclines to omit the balance between queue backlogs, and admit only arriving requests by sending them to the processing queue with least cost. In contrast, if the value of parameter V approaches zero, similar to *GOSCAD*, *POSCAD* would ignore the impact of such costs and greedily forwards requests to the queue with the smallest backlog size. The choice of V reflects the main objective of system design. Last but not least, *POSCAD* can also be run in a distributed fashion with a time complexity of $O(|C|)$. The analysis is similar to that of *GOSCAD* and hence omitted here.



(a) Centralized (b) Decentralized

Fig. 3. Two scheduler implementations for *GOSCAD* and *POSCAD*

V. Simulation Results

We conduct trace-driven simulations to evaluate the performance of *GOSCAD* and *POSCAD* under different settings. First, we introduce the basic simulation settings. Then we compare *GOSCAD* with baseline schemes in terms of the reduction in total costs of communication and computation, as well as the total queue backlog size. Next, we further evaluate *POSCAD* to investigate the benefits of predictive scheduling by conducting analysis on the simulation results.

A. Basic Settings

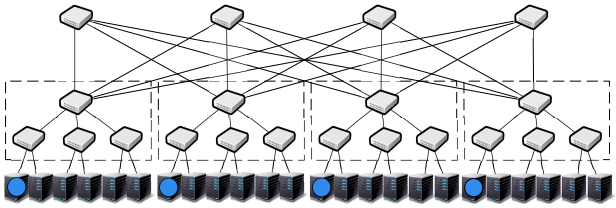
Topology: We conduct extensive simulations under four well-known data center topologies: Canonical 3-Tiered topology [2], Fat-tree [1], Jellyfish [22], and F10 [16]. We show one instance for each of them, respectively, in Figure IV-D. To make our performance analysis comparable among the four topologies, we construct instances of these topologies at almost the same scale, comparable to commercial data centers [2].

We deploy controllers (the control plane of the SDN system) on particular hosts, which are denoted by hosts with blue circles. Specifically, for deterministic topologies (Fat-tree, Canonical 3-Tiered, and F10), we deploy one controller in every pod[6]; for random topology (Jellyfish), we deploy the same number of controllers on hosts with non-adjacent ToRs.

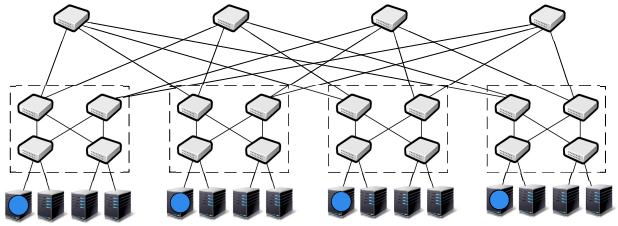
Request Arrival Settings: We conduct trace-driven simulations, where the request arrival process on each switch follows the distribution of request inter-arrival time that are drawn from measurements within real-world data centers [2]. Accordingly, the length of each time slot is set as $10ms$ and average request arrival rate on each switch is about 5.88 requests per time slot. Considering the existence of hot spots in real-world systems, where some switches have intensive request arrivals, in our simulation, we pick the first pod as the hot spot and each switch therein has particularly large request arrival rate (200 requests per time slot). For controllers, each of them has a service capacity of 600 requests per time slot, which is the capacity of a typical NOX controller [24].

System Cost Settings: 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, the average unit computation cost P of switches is set as the average hop number between switches and controllers in the topology. In both Fat-tree and F10 topologies, $P = 4.13$; in 3-Tiered and Jellyfish

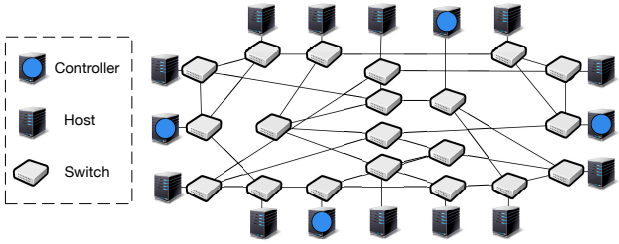
[6]In Canonical 3-Tiered topology, we regard the group of switches that belong to the same aggregation switch as one pod (including the aggregation switch itself).



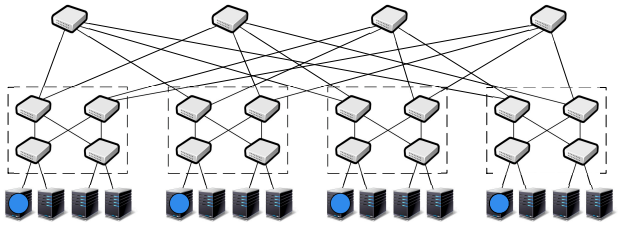
(a) Canonical 3-Tiered with $k = 4$ core switches in the topmost layer, each connecting to k aggregate switches. Each aggregate switch manages $(k-1)$ edge switches, while each edge switch is associated with $k/2$ hosts, and hence $(k^3 - k^2)/2$ hosts in total.



(b) 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.



(c) Jellyfish topology with $k = 4$, where k denotes the number of switch ports. Jellyfish topology is constructed as a random graph at the top-of-rack (ToR) switch layer, with $5k^2/4$ switches and $k^3/4$ hosts in total.



(d) F10 topology ($k = 4$). It has the same numbers of switches and hosts with Fat-tree. The difference lies in that F10 breaks the structural symmetry of Fat-tree with AB-tree design to enhance its fault tolerance.

Fig. 4. Instances of topologies deployed for SDN systems, including Canonical 3-Tiered topology, Fat-tree topology, Jellyfish topology, and F10 topology, respectively. All controllers in the control plane are deployed on particular hosts.

topologies, P is 4.81 and 3.56, respectively. Besides, by setting parameter $\gamma = 1$, we assume the equal importance of reducing communication costs and local computational costs on switches.

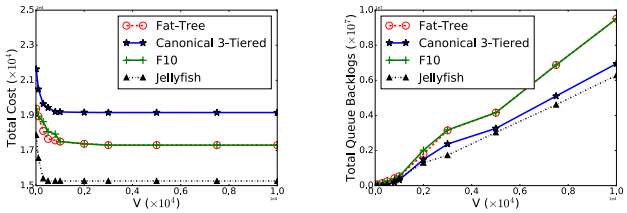
Queueing: Regarding *POSCAD*, we treat all queue backlogs equally by setting weighting factors β_1 and β_2 as 1 for (25). In practice, the value of parameter β_1 and β_2 can be tuned proportional to the ratio between the capacity of prediction queues, switches' queues, and controllers' queues. Besides, recall that the value of parameter V determines the importance of communication cost reduction to queueing 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 costs (in milliseconds). In our simulation, the ratio is about 10^3 and thus we adjust the value of parameter V from 1 to 10^4. The queueing policy is fixed as *first-in-first-out* for all queues.

Scheduler Implementations: Figure 3 (a) and (b) illustrate how *GOSCAD* and *POSCAD* can be implemented in a centralized and decentralized manner, respectively.

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, as well as the unit communication and computation costs, to make a centralized scheduling decision. Next, it spreads the scheduling decision onto switches; then switches upload or locally process their requests according to the decision. We visualize the abstract process in Figure 3 (a). The advantage of centralized architecture is that it requires no modification on the data plane, *i.e.*, all the system dynamics such as the 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 switches' compute resources are scarce. In fact, the scheduler could also be deployed on control plane. However, such a centralized scheduler is a single point of failure, and a bottleneck in face of considerable computations. Besides, it requires back-and-forth message exchange between the SDN system and the scheduler, leading to longer response time.

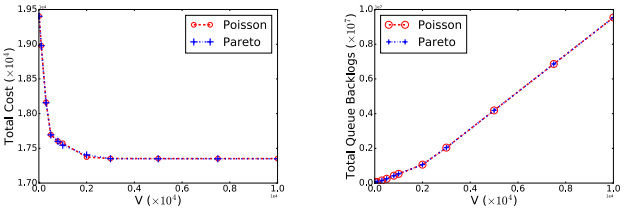
In the decentralized way, as Figure 3 (b) shows, switches periodically update their information about queue backlogs from control plane, while each of them can make scheduling decisions independently. Though this way requires modification on switches, the decentralized way still has the following advantages. It requires less amounts of message exchange than that in the centralized way, thus switches would response even faster to handling network events. On the other hand, the computation of both schemes decision-making is distributed across switches, leading to better scalability and fault tolerance. In our simulation, we choose the decentralized implementation for both *GOSCAD* and *POSCAD*.

Prediction Settings: We vary switches' prediction window sizes by sampling them uniformly at random in $[0, 2\times D]$, with mean D. The value of D ranges 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 all time slots, and generate e_t 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 value. 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



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

Fig. 5. Performance of *GOSCAD* under different topologies.



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

Fig. 6. Performance of *GOSCAD* under Fat-tree topology with other two arrival processes.

variance σ^2. Accordingly, we have

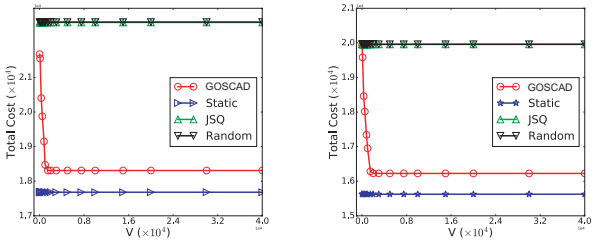
$$r = 2\,[1 - \Phi\,(0.5/\sigma)], \tag{33}$$

where $\Phi(\cdot)$ is the CDF of *standard normal distribution*. Based on (33), the error rate r can be adjusted by choosing an appropriate value of σ. We vary error rate r from 0% to 50%.

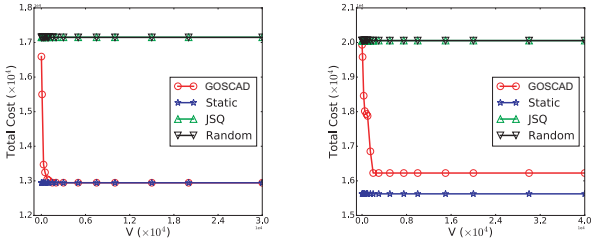
Response Time Metric: As for evaluation metrics, note that the total queue backlog size in the non-prediction case ($D = 0$) and the predictive case ($D > 0$) are incomparable. The reason is that with predictive scheduling, the total queue backlog size, as defined in (25), also includes future untreated requests. All such requests are still counted, although they haven't 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 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 with 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 experience a near-zero response time. The average request response time is obtained over completed requests.

B. Evaluation of GOSCAD

Figure 5 (a) and (b) show how different values of parameter V affect the long-term time-average 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 0 to 1000, 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-average expectation of total costs and total queue backlog size in the system. We discuss them in detail as follows.



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



(c) Jellyfish topology (d) F10 topology

Fig. 7. Comparison of communication costs among four scheduling schemes under different topologies.

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 overly loaded. For switches that are distant from controllers, local processing is a better choice. Consequently, large values of V generally conduces 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*, request response time increases as well.

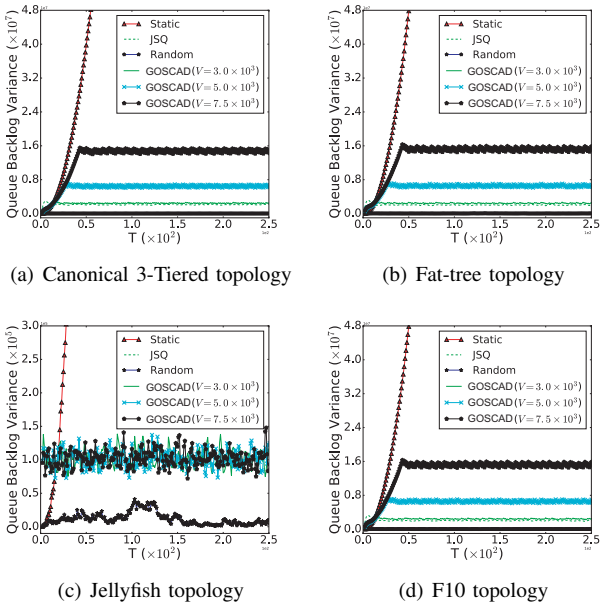
Second, the total costs in 3-Tiered topology is greater than the other schemes', because it has a higher unit computational cost ($\alpha = 4.81$ compared to 4.13 and 3.56 for Fat-tree/F10 and Jellyfish); meanwhile, switches in 3-Tiered topology usually take longer path to controllers and incur more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

Third, the total costs under Jellyfish topology is significantly lower (up to 21%) than the others, as Jellyfish generally has shorter average path between switches and controller than the other deterministic topologies of the same scale [22].

Figure 6 (a) and (b) show the performance of *GOSCAD* under Fat-tree topology with other two request arrival processes. The results show that *GOSCAD* does not require the prior knowledge about the statistics of request arrival dynamics and still achieves the $[O(V), O(1/V)]$ backlog-cost trade-off.

C. Comparison with Other Association Schemes

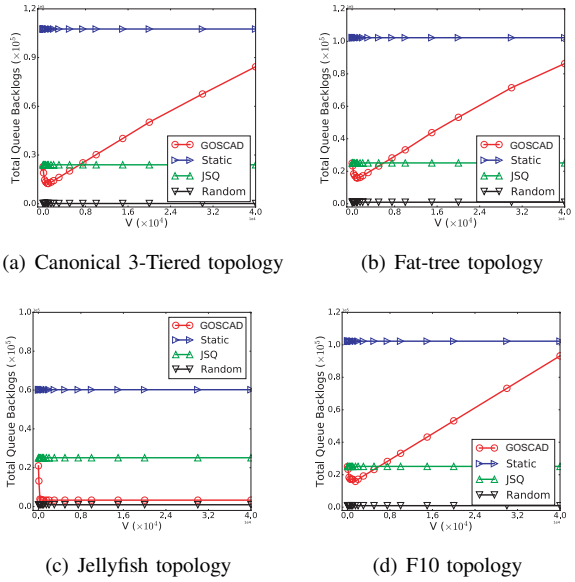
In Figure 7 - 10, we compare *GOSCAD*'s performance with three baseline schemes: *Static*, *Random* and *JSQ (Join-the-Shorest-Queue)*. Particularly, in *Static* scheme, each switch i chooses the controller j with minimum communication costs and then fixes such an association for all time slots. In *Random* scheme, each switch is scheduled to pick up a controller uniformly at random during each time slot. In *JSQ* scheme, each



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

(c) Jellyfish topology (d) F10 topology

Fig. 8. Variance of queue backlog size comparison among four scheduling schemes under different topologies.



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

(c) Jellyfish topology (d) F10 topology

Fig. 9. Total queue backlog size comparison among four scheduling schemes under different topologies.

switch picks the controller with the shortest queue backlog. Considering that the baseline schemes do not conduct control devolution. To make the comparison fair, we set the unit computational cost $\alpha = 10^{28}$ 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 algorithm. Note that such settings also emulate the scenarios when control devolution is not supported.

From Figure 7, 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, which is mainly because their decision making makes 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 queueing 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 backlogs exceed some threshold (about twice the value of V in our simulations). In such cases, *Static* would continue pursuing minimum communication cost 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 cost but smaller queue backlog size, to ensure queueing stability. Such illustration is supported by the results from Figure 8, where we see that the large the value of parameter V, the greater the variance of queue backlog sizes among different controllers. In the extreme case (*Static*), the variance grows explosively. Unlike *Static*, *GOSCAD* constantly ensures queueing 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, and hence more balanced queue backlogs in the system.

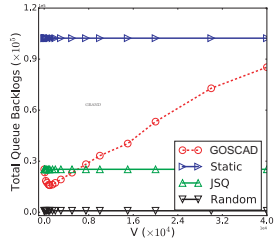
Insight: As the value of V increases, *GOSCAD* achieves a flexible trade-off between system cost reduction and queueing stability. The gap between *GOSCAD* and the minimum costs is the price which has to be taken to maintain queueing stability.

Figure 9 shows the induced total queue backlog sizes under *GOSCAD* and the baseline schemes. We see that under different topologies, *Random* and *JSQ* maintains the total queue backlog sizes at a low level, which is consistent with the result in Figure 8. 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.

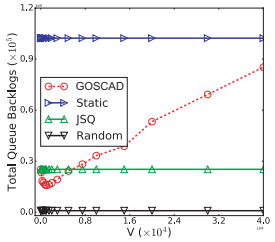
As for *GOSCAD*, under deterministic topologies (3-Tiered, Fat-tree, and F10), we observe a reduction in the backlog size at the very beginning, then a linear increase after reaching a valley at around 1.0×10^3. The explanation is as follows. When the value of V is small, switches prefer controllers with shorter queues. [7] A switch's scheduling decision is independent of the others'. This will lead to arriving requests being intensively uploaded to just few controllers. In this way, controllers close to hot spots are more likely to get 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. Consequently, the skewness of controllers' queue backlogs becomes aggravated and hence the linear rise.

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

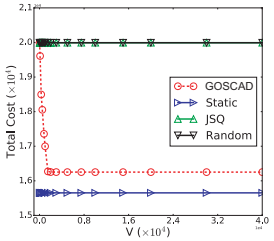
[7] Note that *JSQ* is just a special case of *GOSCAD* with $V = 0$.



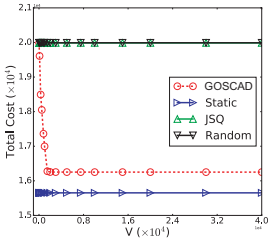
(a) Poisson



(b) Pareto

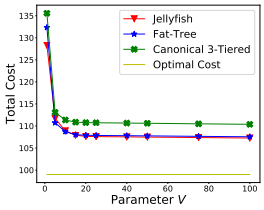


(c) Poisson

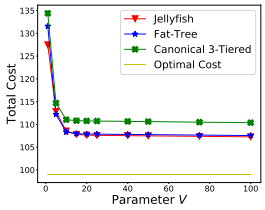


(d) Pareto

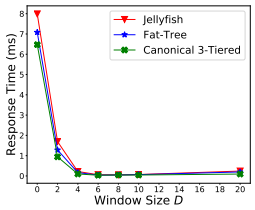
Fig. 10. Performance comparison among different schemes under Fat-tree topology, with Poisson and Pareto request arrivals, respectively.



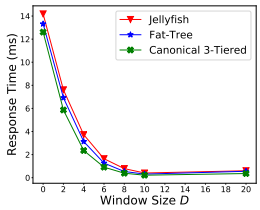
(a) Window Size $D = 0$



(b) Window Size $D = 2$



(c) Trace-driven



(d) Poisson

Fig. 11. Performance under different values of parameter V.

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 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 V has no significant impact on the skewness of controllers' queue backlogs.

Insight: In practice, one can tune the value of parameter V to be around 1 to 1000, to achieve both significant reduction in 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. For Poisson process, we set its arrival rate as 5.88; while for Pareto process, we set its shape parameter as 2 and its scale parameter as 2.94. We only show the simulation results under Fat-tree topology, because the simulation results in other three topologies are qualitatively similar. From Figure 10, we find that the scheduling policies perform qualitatively consistent under different arrival processes.

D. Evaluation of POSCAD

1) 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, *i.e.*, with $r = 0$. Considering the similarities between curves under Fat-Tree and F10 topologies in previous figures. We omit the results regarding F10 in the following.

Time-average total costs vs. V: We compare the total costs incurred by POSCAD with various parameter V among different topologies in Figure 11. In particular, Figure 11 (a) and (b) present the results with a prediction window size $D = 0$ (*non-prediction*) and $D = 2$, respectively. From both plots, we find that as V increases from 1 to 20, all the curves fall rapidly under different topologies. When V continues increasing after 25, the total costs stops decreasing and remains stable thereafter, with a merely 10% gap to the optimal total costs (denoted by the solid horizontal line). Comparing (a) and (b), we observe that the incurred total costs are almost the same even when the window size D is increased. Based on such observations, we have the following insight.

Insight: As the parameter V increases, POSCAD reduces the time-average total costs and achieves the optimal system costs asymptotically under different topologies. Note that the gap between the eventual total costs and the optimal costs is the price paid for balancing and stabilizing queue backlogs in the system. In the mean time, 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 switch to the impact of different prediction window sizes on the average request response time. Figure 11 (c) and (d) present the results drawn from simulations with trace-driven and Poisson arrival process, respectively. Figure 11 (c) shows the curves under the trace-driven setting. We observe a sharp fall of the average request response time from around $14ms$ to less than $1ms$, with the window size rising from 0 (non-prediction) to 6 under different topologies. As the window size continues growing, the reduction stops and the average response time remains stable ($0.45ms$). The results are similar in Figure 11 (d).

Insight: Figure 11 shows the benefits of predictive scheduling. By exploiting predictive information and pre-serving future requests, POSCAD effectively reduces the average response time without inducing extra system cost. Moreover, with mild-value of future information, POSCAD achieves a near-zero average response time.

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

Average response time vs. prediction error rate: Figure 12 shows the average response times with prediction error rate ranging from 0% (perfect prediction) to 50%, under various settings of parameter V and window size D.

First, we focus on the red mesh bars, where parameter $V =$

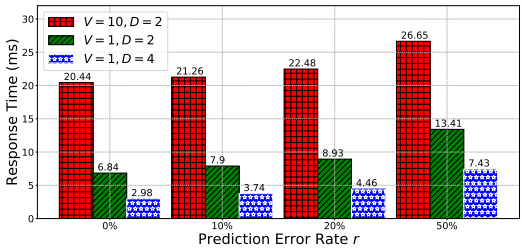


Fig. 12. Average response time vs. prediction error rate under Fat Tree topology with different parameter V and prediction window size D.

10 and prediction 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. Next, with a fixed prediction error rate $r = 10\%$ and a prediction window size $D = 2$, we compare the response times with different values of V. Figure 12 shows that the response time ($21.26ms$) of the red mesh bar ($V = 10$) is higher than that ($7.9ms$) of the green stripped bar ($V = 1$). The result suggests that one can reduce request response time by reducing 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* [15], implies a lower average response time. Nonetheless, reducing V will also increase total costs. Last but not least, by fixing the prediction error rate ($r = 10\%$ as an example), we see the blue bar ($D = 4$) is lower than the green bar ($D = 2$), implying that more future information is helpful to reduce the response time by up to 50% even under inaccurate prediction.

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. 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 *GOSCAD* and *POSCAD*, two efficient joint control schemes that solve the problem through a series of online decision making in a distributed manner. Our theoretical analysis showed that *GOSCAD* achieves near-optimal total costs with queueing stability guarantee. Furthermore, with predictive scheduling, *POSCAD* goes beyond *GOSCAD* by achieving significant reduction in request response time. We conducted extensive simulation results to verify the effectiveness of both proposed schemes. Notably, with mild-value of future information, POSCAD induces a significant reduction in request response time, even in face of mis-prediction.

Appendix A
Problem Transformation by Opportunistically Minimizing an Expectation

By minimizing the upper bound of the drift-plus-penalty expression, [8]

$$\Delta_V(\mathbf{Q}(t)) \triangleq \Delta(\mathbf{Q}(t)) + V \cdot E\left\{f_{\mathbf{X}}(t) + g_{\mathbf{X}}(t)|\mathbf{Q}(t)\right\}, \tag{34}$$

[8]Recall that V is a positive constant parameter that weighs the penalty brought by communication costs between switches and controllers ($f_{\mathbf{X}}(t)$) and the local computation costs on switches due to control devolution ($g_{\mathbf{X}}(t)$).

the time average of communication cost can be minimized while stabilizing the network of request queues. We denote the objective function to be solved in time slot t by $J_t(\mathbf{X})$, *i.e.*,

$$\begin{aligned} J_t(\mathbf{X}) \triangleq \sum_{i\in\mathcal{S}}\Big[V\cdot P_i(t) + Q_i^s(t)\Big]A_i(t) + \sum_{i\in\mathcal{S}}\sum_{j\in C}\Big[VM_{i,j}(t) \\ +Q_j^c(t) - VP_i(t) - Q_i^s(t)\Big]\mathbf{X}_{i,j}A_i(t) \end{aligned} \tag{35}$$

and its optimal solution by $\mathbf{X}^* \in \mathcal{A}$.

Therefore, for any other scheduling decision $\mathbf{X} \in \mathcal{A}$ made during time slot t, we have

$$J_t(\mathbf{X}) \geq J_t(\mathbf{X}^*). \tag{36}$$

Taking the conditional expectation on both sides conditional on $\mathbf{Q}^c(t)$, we have

$$E\left[J_t(\mathbf{X}) \,|\, \mathbf{Q}^c(t)\right] \geq E\left[J_t(\mathbf{X}^*) \,|\, \mathbf{Q}^c(t)\right], \tag{37}$$

for any $\mathbf{X} \in \mathcal{A}$. In such a way, instead of directly solving the long-term stochastic optimization problem, we can opportunistically choose a feasible association that minimizes (35) during each time slot.

Appendix B
Proof of the $[O(V), O(1/V)]$ Trade-off

We assume there is an S-only algorithm achieves optimal time-average total costs (infimum) $\bar{f}^* + \gamma\bar{g}^*$ with action $\mathbf{X}^*(t)$ and $\mathbf{Y}^*(t)$, $t = \{0, 1, 2, \cdots\}$ [20]. With the existence of randomized scheduling scheme which ensures that the expectation of total system processing capacity to be strictly greater than that of the total request arrival rate, *i.e.*, there exists some $\epsilon \geq 0$ such that

$$\mathbb{E}\left\{\sum_{i\in\mathcal{S}} X_i^*(t)\Big|\mathbf{Q}(t)\right\} + \epsilon \leq \mathbb{E}\left\{\sum_{j\in C} B_j^c(t) + \sum_{i\in\mathcal{S}} B_i^s(t)\Big|\mathbf{Q}(t)\right\}. \tag{38}$$

Next, we denote $\mathbf{X}'(\mathbf{t})$, $\mathbf{Y}'(\mathbf{t})$ as the decisions over time, and $f'(t)$, $g'(t)$ as the corresponding costs given by POSCAD algorithm. Accordingly, the one-slot drift-plus-penalty is

$$\begin{aligned} \Delta_V(\mathbf{Q}(t)) \leq\ & B + C(\mathbf{Q}(t)) + \\ & \mathbb{E}\left\{\sum_{j\in C} Q_j^c(t)\left(\sum_{i\in\mathcal{S}} X_i^*(t)Y_{i,j}^*(t)\right)\Big|\mathbf{Q}(t)\right\} \\ & +\alpha\mathbb{E}\left\{\sum_{i\in\mathcal{S}} Q_i^s(t)\left(1 - \sum_{j\in C} Y_{i,j}^*(t)\right)X_i^*(t)\Big|\mathbf{Q}(t)\right\} \\ & -\beta\mathbb{E}\left\{\sum_{i\in\mathcal{S}} Q_i^p(t)X_i^*(t)\Big|\mathbf{Q}(t)\right\} + \\ & V\mathbb{E}\left\{f^*(t) + \gamma g^*(t)\Big|\mathbf{Q}(t)\right\}. \end{aligned} \tag{39}$$

Using (38) and the definition in (25), we obtain

$$\begin{aligned} &\mathbb{E}\left\{L(t+1) - L(t)\Big|\mathbf{Q}(t)\right\} + V\mathbb{E}\left\{f'(t) + \gamma g'(t)\Big|\mathbf{Q}(t)\right\} \\ &\leq B + V\mathbb{E}\left\{f^*(t) + \gamma g^*(t)\Big|\mathbf{Q}(t)\right\} - \epsilon\mathbb{E}\left\{h(t)\Big|\mathbf{Q}(t)\right\}, \end{aligned} \tag{40}$$

where $L(t) \triangleq L(\mathbf{Q}(t))$. Then taking expectation over both sides and summing over $t \in \{0, 1, \cdots, T-1\}$, we have

$$\mathbb{E}\left\{L(T-1)\right\} - \mathbb{E}\left\{L(0)\right\} + V\mathbb{E}\left\{\sum_{t=0}^{T-1}\left(f'(t) + \gamma g'(t)\right)\right\}$$

$$\leq BT + V\mathbb{E}\left\{\sum_{t=0}^{T-1}\left(f^*(t)+\gamma g^*(t)\right)\right\} - \epsilon\cdot\mathbb{E}\left\{\sum_{t=0}^{T-1}h(t)\right\}. \quad (41)$$

We are ready to prove the $[O(V), O(1/V)]$ trade-off.

1) First, dividing both sides of (41) by VT, rearranging items, and then canceling non-positive quantities, we obtain

$$\begin{aligned}&\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\{\left(f'(t)+\gamma g'(t)\right)\right\}\\ &\leq \frac{B}{V}+\frac{\mathbb{E}\{L(0)\}}{VT}+\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\{\left(f^*(t)+\gamma g^*(t)\right)\right\}.\end{aligned} \quad (42)$$

Taking lim-sup as $T\to\infty$ in (42) yields

$$\begin{aligned}&\limsup_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\{\left(f'(t)+\gamma g'(t)\right)\right\}\\ &\qquad\leq \limsup_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\{\left(f^*(t)+\gamma g^*(t)\right)\right\}+\frac{B}{V}.\end{aligned} \quad (43)$$

2) Similarly, by dividing both sides of (41) by ϵT, we have

$$\begin{aligned}&\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\{h(t)\right\}\leq\\ &\frac{B}{\epsilon}+\frac{\mathbb{E}\{L(0)\}}{\epsilon T}+\frac{V\sum_{t=0}^{T-1}\mathbb{E}\left\{\left(f^*(t)+\gamma g^*(t)\right)\right\}}{\epsilon T}.\end{aligned} \quad (44)$$

Since $L(0)$ is constant, by taking lim-sup as $T\to\infty$, we obtain

$$\limsup_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\{h(t)\right\}\leq\frac{V}{\epsilon}\left(\bar{f}^*+\gamma\bar{g}^*\right)+\frac{B}{\epsilon}<\infty. \quad (45)$$

■

Appendix C
Motivating Example of Predictive Scheduling

Figure 13 presents an example that compares the cases with and without predictive scheduling, where Figure 13(a) shows the system state at the beginning of time slot t. Wherein there are two switches potentially connected to one controller through dashed arrows. All requests are assumed homogeneous and can be handled by both switches and the controller. Upon new requests' arrival, each switch either associates with the controller through a solid arrow and uploads requests, or stores them in its own queue for local processing. In each time slot, a switch processes at most 1 request and the controller processes at most 2 requests, both in a FIFO manner. The objective is to minimize the average request response time. (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. (b) and (c) present the scheduling process that handles current requests only, whereas (d) and (e) exhibit the case with predictive scheduling.

First, we consider the case without predictive scheduling and handle only the arriving requests, as shown in Figure 13 (b) and (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 Figure 13 (b), the controller and switches process the requests in their respective queues. Figure 13 (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.

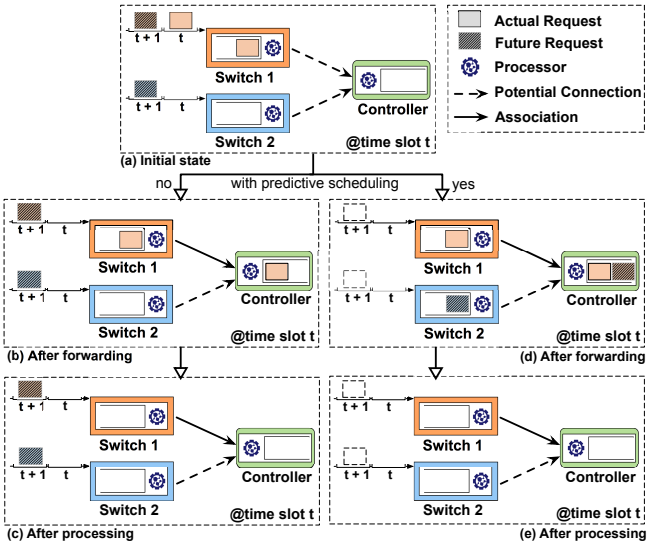


Fig. 13. An example that shows the benefits of predictive scheduling.

Next, we focus on the case with predictive scheduling, as shown in Figure 13 (d) and (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 $(t+1)$, then uploads the request to the controller. Similarly, switch 2 pre-admits the future request in time $(t+1)$ and stores the request in its local queue. Figure 13 (d) shows the system state after the (pre-)admission of requests. Then the controller and switches consume the requests from their queues. Figure 13 (e) shows that with predictive scheduling, all the requests in time t and $(t+1)$ are completed by the end of time t. Consequently, both future requests will receive instant response upon their arrivals in time $(t+1)$.

The above example shows that predictive scheduling can effectively reduce the request response time by taking advantages of utilizing the present surplus processor capacities. In the next subsection, we extend our previous model and formulation in Section **??** with predictive scheduling.

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of ACM SIGCOMM*, 2008.
[2] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of ACM IMC*, 2010.
[3] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*. John Wiley & Sons, 2015.
[4] J. Broughton, "Netflix adds download functionality," https://technology.ihs.com/586280/netflix-adds-download-support, 2016.
[5] 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 Performance Evaluation Review*, vol. 44, no. 1, pp. 193–206, 2016.
[6] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: scaling flow management for high-performance networks," in *Proceedings of ACM SIGCOMM*, 2011.
[7] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," in *Proceedings of ACM HotSDN*, 2013.

[8] J. Dunn, "Introducing FBLearner Flow: Facebook's AI backbone," https://code.fb.com/ml-applications/introducing-fblearner-flow-facebook-s-ai-backbone/, 2016.

[9] A. Filali, A. Kobbane, M. Elmachkour, and S. Cherkaoui, "Sdn controller assignment and load balancing with minimum quota of processing capacity," in *Proceedings of IEEE ICC*, 2018.

[10] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proceedings of ACM HotSDN*, 2012.

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

[12] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks." in *Proceedings of USENIX OSDI*, 2010.

[13] A. Krishnamurthy, S. P. Chandrabose, and A. Gember-Jacobson, "Pratyaastha: An efficient elastic distributed sdn control plane," in *Proceedings of ACM HotSDN*, 2014.

[14] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: state distribution trade-offs in software defined networks," in *Proceedings of ACM HotSDN*, 2012.

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

[16] V. Liu, D. Halperin, A. Krishnamurthy, and T. E. Anderson, "F10: A fault-tolerant engineered network." in *Proceedings of USENIX NSDI*, 2013.

[17] X. Lyu, C. Ren, W. Ni, H. Tian, R. P. Liu, and Y. J. Guo, "Multi-timescale decentralized online orchestration of software-defined networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 12, pp. 2716–2730, 2018.

[18] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[19] S. Nanda, F. Zafari, C. DeCusatis, E. Wedaa, and B. Yang, "Predicting network attack patterns in sdn using machine learning approach," in *Proceedings of IEEE NFV-SDN*, 2016.

[20] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[21] I. S. Petrov, "Mathematical model for predicting forwarding rule counter values in sdn," in *Proceedings of IEEE Conference of EIConRus*, 2018.

[22] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers, randomly." in *Proceedings of USENIX NSDI*, 2012.

[23] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of Internet Network Management Conference on Research on Enterprise Networking*, 2010.

[24] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proceedings of ACM Hot-ICE*, 2012.

[25] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic sdn controller assignment in data center networks: Stable matching with transfers," in *Proceedings of IEEE INFOCOM*, 2016.

[26] T. Wang, F. Liu, and H. Xu, "An efficient online algorithm for dynamic sdn controller assignment in data center networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2788–2801, 2017.

[27] J. Yang, X. Yang, Z. Zhou, X. Wu, T. Benson, and C. Hu, "Focus: Function offloading from a controller to utilize switch power," in *Proceedings of IEEE NFV-SDN*, 2016.

[28] H. Yu, M. H. Cheung, L. Huang, and J. Huang, "Power-delay tradeoff with predictive scheduling in integrated cellular and wi-fi networks," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 4, pp. 735–742, 2016.

[29] S. Zhang, L. Huang, M. Chen, and X. Liu, "Proactive serving decreases user delay exponentially," *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 2, pp. 39–41, 2015.

[30] K. Zheng, L. Wang, B. Yang, Y. Sun, Y. Zhang, and S. Uhlig, "Lazyctrl: Scalable network control for cloud data centers," in *Proceedings of IEEE ICDCS*, 2015.