Online VNF Chaining and Scheduling with Prediction: Optimality and Trade-offs

Xi Huang, Simeng Bian, Xin Gao, Weijie Wu, Ziyu Shao, Yang Yang
School of Information Science and Technology, ShanghaiTech University
Email: {huangxi, biansm, gaoxin, shaozy, yangyang}@shanghaitech.edu.cn, wuwjpku@gmail.com

***Abstract*—For NFV systems, the key design space includes the function chaining for network requests and the resource scheduling for servers. The problem is challenging since NFV systems usually require multiple (often conflicting) design objectives and the computational efficiency of decision making with limited information. Besides, the limits and benefits of predictive scheduling to NFV systems still remain unexplored.**

In this paper, we propose *POSCARS*, an efficient, distributed and online algorithm that achieves a tunable trade-off between various system metrics with stability guarantee, while exploiting the power of predictive scheduling. With a careful choice of granularity in problem formulation and system modeling, we acquire a deeper understanding of the trade-offs in our design space. By a non-trivial transformation, we decouple the complex optimization problem into a series of online sub-problems that achieve optimality with only limited information. Leveraging sliding lookahead window techniques, we further improve the system performance via predictive scheduling. Using randomized load balancing techniques, we propose three variants of POSCARS to further reduce sampling overheads. Theoretical analysis and trace-driven simulations show that POSCARS and its variants require only mild-value of future information to achieve a near-minimum average system cost while effectively shortening the average request response time.

I. Introduction

Network function virtualization (NFV) is shifting the way of network service deployment and delivery by virtualizing and scaling network functions (NFs) on commodity servers in an on-demand fashion [1]. As a revolutionary technique, NFV paves the way for operators towards better manageability, reliability, and performance guarantee on network services.

Typically, in a NFV system, a network service is implemented in a form of an ordered chain of virtual network functions (VNFs) that are deployed on commodity servers, *a.k.a.* a service chain. Along the chain, every VNF performs some particular treatment on the received requests, then hands over the output to the next VNF in a pipeline fashion. To enable a network service, one needs to *place*, *activate*, and *chain* VNFs deployed on various servers.

Due to the high cost of VNF migration and instantiation [2], VNF replacement can only be performed infrequently; that being said when considering flow- or request-scale operations, function placement can be viewed as static. Given this fact, a natural practice is to place multiple VNFs in one server in advance, but due to hardware resource constraints (*e.g.*, CPU, memory, and storage), a server must carefully schedule resources among a *subset* of such VNFs at a particular time (*i.e.*, only a subset of VNF instances can be activated on a server at a particular time). Therefore, with a fixed VNF placement, the activation and chaining of VNFs refer to: 1) for each server, the *resource allocation* to a subset of deployed VNFs subject to resource constraints; and 2) for each network service, the *selection* of the activated instances for its VNFs, so as to determine the sequence of instances that the requests will visit, *a.k.a. service chaining*.

Given that VNF placement is considered static at the time scale of flow or request operations, for service chaining and resource scheduling, a natural question is: should they also be static, or dynamic? Static schemes have been implemented in some scenarios, but often times request traffic is highly variable in both temporal and spatial dimensions [3]. In such cases, static schemes may lead to imbalanced workloads among instances, leaving a specific instance overloaded or under-utilized. Hence, there is a huge demand to design an efficient and dynamic scheme that performs service chaining and resource scheduling, which adapts to traffic variations and achieves load balancing in real time. As for implementability, recent advances (e.g., temporal and spatial processor sharing [4]) have enabled servers to adjust resource allocation among various functions in real time.

Such dynamic design is non-trivial, especially given the complex interplay between instances of successive VNFs and the resource contention among instances on servers. In particular, we would like to address the following challenges:

Characterization of the tunable trade-offs among various performance metrics: NFV systems often have multiple optimization objectives, *e.g.*, maximizing resource utilization, minimizing energy consumption, and reducing request response time. Different stakeholders may have different preferences over these objectives, and often times they conflict with each other. It is important to characterize their trade-offs, so as to acquire a fundamental understanding of our design space and tune the system towards the particular state that we desire.

Efficient online decision making: VNF request processing often requires low latency and high throughput. Hence, an effective dynamic scheme must also be computationally efficient, and can be adaptive to request changes. This is challenging not only because of the nature of the high complexity, but also that service requests arrive in an online manner, while the underlying traffic statistics are often unknown a priori. All these uncertainties make it more challenging to optimize system objectives through a series of online decisions, not to mention that we prefer a distributed manner.

Understanding benefits of predictive scheduling: A natural optimization of online decision making is to consider prediction of future information to reduce response time and improve service quality via learning techniques. However, there is no free lunch. For example, Netflix preloads videos onto users' devices based on user behavior prediction; such a preloading can be wasteful if wrongly decided [5]. Despite the wide applications of such prediction-based approaches [6]–[8], it still remains open to what extent a predictive scheduling can be beneficial to NFV systems, even in the presence of prediction errors. Answers to the questions are the key to

understanding the endeavor worthy to put on VNF scheduling prediction, and whether one can tolerate the worst possible case that may occur.

Despite the number of recent works on VNF scheduling, as far as we are aware, there is still no fundamental understanding on the above questions, nor is there any strategy that can achieve the design objectives simultaneously in a fully online fashion. One important reason is in the difficulty of problem formulation and modeling, especially in choosing the granularity. If one models the system state and strategy in flow-level abstraction [9], it may fall short in accurate characterization of interplay between successive VNF instances and system dynamics over time; however, if one applies fine-grained control to each request [6], then the decision making will inevitably incur a rather high computational overhead. Such issue does not only prohibit a deep understanding on system dynamics, but also prevent us from obtaining efficient and accurate strategy design.

In this paper, we overcome such difficulty by applying a number of novel techniques. Our contributions are:

Model and formulation: We propose a novel model that separates the granularity of system state characterization and strategy making. In particular, we use a queuing model at the request granularity to characterize system dynamics. Unlike flow-level abstraction, our model require no prior knowledge on underlying flows, but accurately captures the interplay between successive instances, *i.e.*, real-time dynamics of how requests are received, processed, and forwarded. In contrast, our decision making is at the granularity of request batch in a per-time-slot manner to avoid high overheads. Such a careful choice makes it possible to characterize the system dynamics and performance in a clear yet accurate way.

Algorithm design: To enable online and efficient decision making, we transform the long-term stochastic optimization problem into a series of sub-problems over time slots. By exploiting their unique structure, we propose *POSCARS* (Predictive Online Service Chaining And Resource Scheduling scheme), with provably asymptotic optimality that achieves near-optimal system cost and a tunable trade-off among various system objectives in a distributed manner.

Predictive scheduling: To the best of our knowledge, this paper is the first to address the dynamic service chaining and scheduling problem in NFV system by jointly considering resource utilization, energy efficiency, and response latency. This paper is also the first to study the fundamental benefits of predictive scheduling with future information in NFV system. We extend a new dimension for NFV system design by proposing predictive scheduling.

Experiment verification and investigation: We conduct trace-driven simulations and results show the effectiveness of POSCARS and its variants under various settings against baseline schemes, as well as the benefits of predictive scheduling in achieving ultra-low request response time.

The rest of this paper is organized as follows. In Section II, we present our model and formulation. In Section III, we present the design of POSCARS and its variants, followed by the corresponding performance analysis. We show the simulation results and analysis in Section IV, review related work in Section V, and conclude the paper in section VI.

II. Motivating Example

In this section, we show a motivating example that exhibits the potential trade-off in the multi-objective optimization for different system metrics, including reduction in energy cost and communication cost, as well as shortening response times, which is mainly due to queueing delay. Besides, the example also explores the value of future information and the potential benefit of predictive scheduling.

We consider a time slotted NFV system, where predictive scheduling is viable, *i.e.*, the request in time $(t+1)$ can be perfectly predicted and pre-served by the system. Figure 1 (a) shows the basic settings and initial system state in time t. All VNF instances are readily deployed on servers in a fixed manner. Each instance maintains a queue backlog to buffer untreated requests. Every server has a service capacity of two requests per time slot. Processing a request incurs an energy cost of 1. Note that 1) any requests processed by VNF a's instance is not counted in queue backlogs, but readily to be sent to VNF b's instances in the next time slot; 2) all requests processed by VNF b's instances are considered finished.

In this case, there are two possible service chaining decisions, *i.e.*, forwarding the processed request from the instance of VNF a to either VNF b's instance on server II (Decision #1) or server III (Decision #2). It takes a communication cost of 1 to forward the request to VNF b's instance on server II. The communication cost is 2 to the other instance of VNF b on server III.

Our goal is to choose a service chaining decision in time t that jointly minimizes the total energy cost, total communication cost, and the total residual backlog size at the end of time t. [1] Figure 1 (b) - (d) compare the scheduling processes under different service chaining decisions.

In Fig. 1 (b), the new request in time t is admitted, while the processed request is forwarded to the instance of VNF b on server II. Although incurring a low communication cost of 1, such a decision also leads to imbalanced queue backlogs on VNF b's instances. Remind that every server can serve at most two requests per time slot. Hence, servers will then process four requests in total, including the new request on server I, two requests on server II, and another one on server III. The processing incurs a total energy cost of 4. After processing, VNF b's instance on server II still has one untreated request in its backlog. Thus Decision #1 *incurs a total cost of* 5 *on energy and communication, with a residual backlog size of* 1.

On the other hand, when the processed request is forwarded to the instance of VNF b on server III, the decision incurs a high communication cost of 2 but results in balanced queue backlogs on VNF b's instances. Servers will process five requests in total, including the new request on server I, and the rest from server II and III. The processing incurs a total energy cost of 5. After processing, there are no untreated request left in the backlogs. Thus Decision #2 incurs *a higher total cost of* 7 *on energy and communication, but with no residual backlogs*.

Insight 1: Figure 1 (b) and (c) show that we can't achieve the optimal values for different system metrics simultaneously, *i.e.*, there is a potential trade-off between optimizing the total system cost and reducing the total queue backlog size.

[1] By applying *Little's law*, a small queue backlog implies short queueing delay or short response time.

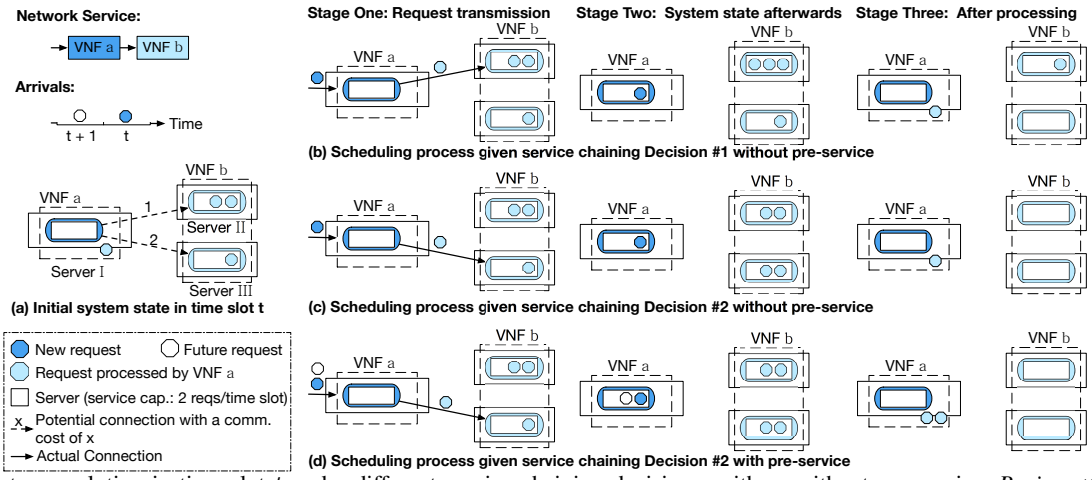


Fig. 1: The system evolution in time slot t under different service chaining decisions, with or without pre-service. *Basic settings:* There is one network service with two VNFs, *i.e.*, VNF a and VNF b. VNF a has one instance, while VNF b has two instances. Every instance maintains one queue backlog to buffer untreated requests. All the instances are readily deployed, with VNF a's instance on server I, while the instances of VNF b on server II and III, respectively. VNF a's instance is potentially connected to both instances of VNF b. *Initial state:* one new request has arrived at time t and another one will arrive at time $(t+1)$. Besides, VNF a's instance has one request that has been processed in time $(t-1)$ and to be sent to one of VNF b's instances in time t.

Additionally, we find that server I is under-utilized in both Fig. 1 (b) and (c), because VNF a's instance only receives and handles the new request at time t. In fact, Fig. 1 (d) shows that we can exploit the spare processing power on server I by pre-admitting and pre-serving the future request. Consequently, we can shorten the response time for the future request by incurring one more energy cost in time t. Note that pre-service does not introduce extra energy cost but actually pays it beforehand. The reason is that even without pre-service, we still have to pay one energy cost in the subsequent time slots after the future request arrives.

Insight 2: By utilizing servers' spare processing power and paying system costs in advance, predictive scheduling can effectively shorten response times of future requests.

To characterize the non-trivial trade-off and exploit the power of predictive scheduling in NFV systems, we present our formulation in the next section.

III. Problem Formulation

We consider a time slotted NFV system, where virtualized network functions (VNF) are instantiated, deployed over a substrate network, and chained together to deliver numbers of network services. Upon the arrival of new network service requests, each VNF processes and hands over requests to its following VNF in a pipeline fashion. All requests are assumed homogeneous. Note that our model can be easily extended to the case with heterogeneous requests. We show an instance of our system model in Figure 2.

A. Substrate Network Model

We consider the substrate network of the system with a set $\mathcal{S}$ of heterogeneous servers. On each server s, we consider R types of resources, *e.g.*, GPU [10], CPU cores [4], and cache [11]. The i-th resource type has a capacity of $c_{s,i}$ and a unit cost of $\lambda_{s,i}$. We denote the resource capacity vector $[c_{s,i}]_{i=1}^{R}$ by $\boldsymbol{c}_s$, and the resource unit cost vector $[\lambda_{s,i}]_{i=1}^{R}$ by $\boldsymbol{\lambda}_s$.

For every server pair (s', s), we use $w_{s',s}(t)$ to denote the communication cost of transferring a request between the

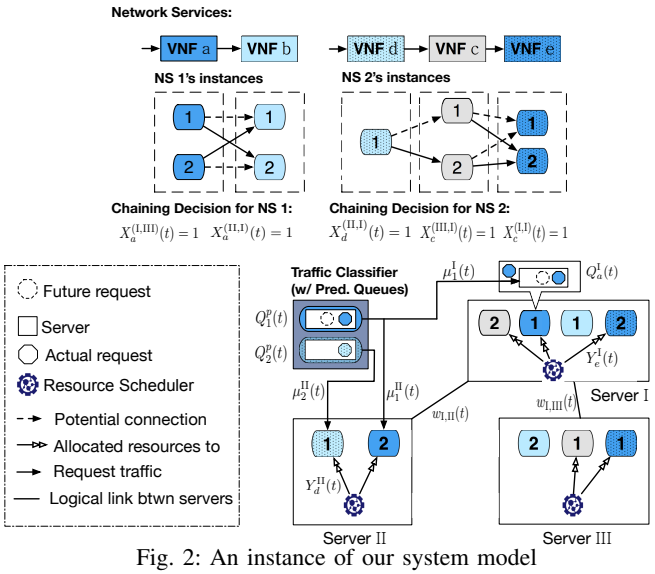


Fig. 2: An instance of our system model

servers in time t, *e.g.*, the number of hops or round-trip times. If two servers are not reachable from each other in time t, then we set $w_{s',s}(t) = +\infty$. The set of all communication costs $[w_{s',s}(t)]_{s',s}$ in time slot t is denoted by $\boldsymbol{w}(t)$.

B. Network Service Model

There are K network services and a set $\mathcal{F}$ of VNFs. Each network service k is represented by a chain of L_k ordered VNFs, wherein the j-th VNF is denoted by $f_{k,j}$. To avoid triviality, we assume that $L_k \geq 2$ for every network service k. Note that L_k is a constant and usually not very large [12]. We regard the same VNF that appears in different service chains as distinct VNFs. In practice, one can set up multiple queues on one VNF instance to buffer requests for different services and map each queue to one VNF instance in our model.

Next, we use $\mathcal{F}_{in}$ to denote $\{f_{k,1}\}_{k=1}^{K}$, *i.e.*, the set of ingress VNFs of all network services, and $\mathcal{F}_{nt}$ to denote the set of non-terminal VNFs of all network services. For every VNF

$f \in \mathcal{F}$, we denote its network service by k_f. If $f \notin \mathcal{F}_{in}$, *i.e.*, not the first VNF of its network service, then we denote its previous VNF by $p(f)$; likewise, if $f \in \mathcal{F}_{nt}$, *i.e.*, not a terminal VNF, then we denote its next VNF by $n(f)$.

C. Deployment Model

In practice, due to request workload changes, it's common to provide multiple instances for every VNF, encapsulate the instances into containers, and distribute them on servers for better load balancing and fault tolerance [13]. In our model, we assume that a VNF may have more than one instance that reside on different servers. A server s hosts a subset $\mathcal{F}_s \subseteq \mathcal{F}$ of VNFs; for every VNF, the server can host at most one instance of that VNF. Note that our model can be further extended to the case when a server hosts multiple instances of the same VNF. Next, we assume that the placement of all VNF instances are pre-determined by adopting VNF placement schemes similar to existing ones [14]–[17].

For VNF $f \in \mathcal{F}$, we use $\mathcal{S}_f$ to denote the set of servers that host f's instances. Every instance maintains one queue backlog to buffer its relevant requests. For example, if VNF f has one instance on server s, then the instance has a queue backlog of size $Q_f^s(t)$ at the beginning of time slot t. Instead of individual queues, one can also implement a shared public queue backlog among instances of the same VNF. All requests from preceding VNF's instances are firstly forwarded and buffered in the public queue. These buffered requests are then rescheduled to one or more idle or least loaded instances. Such a way brings more flexibility so that requests can avoid the potential long queueing delay on individual instances. However, it requires additional physical storage and communication cost due to additional rescheduling. The choice depends on the trade-off made by system designers. Here we adopt the queue model for each individual instance.

D. Predictive Request Arrival Model

For network service k, we use $A_k(t)$ ($\leq a_{max}$ for some constant a_{max}) to denote the number of its new requests that arrive in time slot t, and independent over time slots. In practice, considering the statefulness of VNFs, requests may be aggregated and scheduled in the unit of flow. Our model captures the system dynamics at a finer granularity than the flow-level abstraction, and can be further extended to the case with correlation between requests.

Next, we consider a system which can predict and preserve future request arrivals for network services in a finite number of time slots ahead. Though the technique and analysis of prediction is still under active development [6]–[8], we do not assume any particular prediction technique in this paper. Instead, we assume the prediction as the output from other standalone predictive modules, and investigate the *fundamental* benefits by acquiring and leveraging such future information and the risks induced by mis-prediction. Note that such an assumption is valid to approximate practical scenarios where short-term prediction is viable. For example, Netflix promotes its quality-of-experience (QoE) by predicting user demand and network conditions, then prefetching video frames onto user devices [5].

We assume that for network service k, the system has perfect access to its future requests in a prediction window of size D_k ($\leq D$ for some constant D), denoted by $\{A_k(t+1), \ldots, A_k(t+D_k)\}$. In practice, however, such prediction may be error-prone; we shall evaluate the impact of mis-prediction in the simulation. With pre-service, some future requests may have been admitted into or even pre-served before time t, thus we use $Q_k^{(d)}(t)$ $(0 \leq d \leq D_k)$ to denote the number of untreated requests in slot $(t+d)$ at time t, such that

$$0 \leq Q_k^{(d)}(t) \leq A_k(t+d). \tag{1}$$

Note that $Q_k^{(0)}(t)$ denotes the number of untreated requests that arrive at time t. Therefore, the total number of untreated requests for service k is $Q_k^p(t) = \sum_{d=0}^{D_k} Q_k^{(d)}(t)$. Here we can treat $Q_k^p(t)$ as a virtual prediction queue backlog that buffers untreated future requests for network service k. In practice, the prediction queue backlogs can be hosted on servers or storage systems in proximity to the request traffic classifier [18]. To simplify notations, we use $\boldsymbol{Q}(t)$ to denote the vector of all queue backlogs $\{Q_k^p(t)\}_{k=1}^K$ and $\{Q_f^s(t)\}_{s\in\mathcal{S}, f\in\mathcal{F}_s}$.

E. System Workflow and Scheduling Decisions

Before specifying the scheduling decisions, we first elaborate the overall system workflow in each time slot t.

System Workflow: At the beginning of time slot t, system components (including traffic classifier, VNF instances, and servers) collect relevant system dynamics to decide request admission, service chaining, and resource allocation. According to the decisions, the traffic classifier admits new requests for different network services. VNF instances steer the requests which are processed in time slot $(t-1)$ to their next VNF's instances. Meanwhile, every server allocates the resources to its resident VNF instances [4]. The instances then process the requests from their respective queues. At the end of time slot t, each prediction window moves one slot ahead.

Thus we have three kinds of scheduling decisions to make, *i.e.*, admission, chaining, and resource allocation, respectively.

Admission Decision: For every network service, the traffic classifier decides the number of untreated newly arriving and future requests, to admit into the system. Particularly, for a network service k and its respective ingress VNF f, the classifier decides $\mu_k^s(t)$, *i.e.*, the number of admitted requests to f's instance on server $s \in \mathcal{S}_f$. We use $\delta_k(t)$ to denote the total number of admitted requests from prediction queue $Q_k^p(t)$. These admitted requests should include at least all the untreated requests that actually arrive, while not exceeding $Q_k^p(t)$, *i.e.*, in time slot t and for $k = 1, \ldots, K$,

$$Q_k^{(0)}(t) \leq \delta_k(t) \triangleq \sum_{s\in\mathcal{S}_{f_{k,1}}} \mu_k^s(t) \leq Q_k^p(t). \tag{2}$$

Note that requests are admitted in a fully-efficient manner [19]. In other words, by admitting $\delta_k^{(d)}(t)$ untreated requests from $Q_k^{(d)}(t)$ for $0 \leq d \leq D_k$, the allocation should ensure a total number of $\delta_k(t)$ requests to be admitted, *i.e.*,

$$\sum_{d=0}^{D_k} \delta_k^{(d)}(t) = \delta_k(t) \ \ \forall k \in \{1, \ldots, K\}. \tag{3}$$

The untreated request backlog $Q_k^{(d)}(t)$ evolves as follows,

$$Q_k^{(d)}(t+1) = \left[Q_k^{(d+1)}(t) - \delta_k^{(d+1)}(t)\right]^+, \ \forall\ d \in [0, D_k - 1]. \tag{4}$$

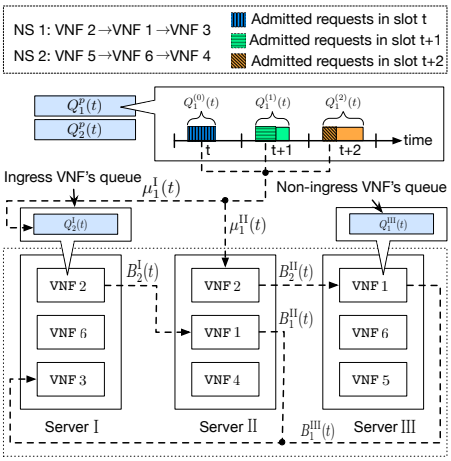


Fig. 3: An instance of system queue model with a lookahead window size of 2. The figure shows how requests are admitted and transferred between successive queues for the first network service (NS 1) in time t, given admission decision $\mu_1^{\mathrm{I}}(t)$, $\mu_1^{\mathrm{II}}(t)$, and chaining decision $X_2^{(\mathrm{I,II})} = X_2^{(\mathrm{II,III})} = X_1^{(\mathrm{II,I})} = X_1^{(\mathrm{III,I})} = 1$.

while $Q_k^{(D_k)}(t+1) = A_k(t+D_k+1)$, where we define $[x]^+ \triangleq \max\{x, 0\}$. We denote all admission decisions by $\boldsymbol{\mu}(t)$.

Service Chaining Decision: Given a non-terminal VNF f, we denote $X_f^{(s',s)}(t) \in \{0,1\}$ as the service chaining decision at time t. We consider the case when VNF f and its next VNF $n(f)$ have instances on server s' and s, respectively. The decision with value 1 indicates the processed requests from VNF f's instance on server s' will be sent to $n(f)$'s instance on server s, and 0 otherwise. To ensure that every instance has a target instance to send its requests, we have

$$\sum_{s \in \mathcal{S}_{n(f)}} X_f^{(s',s)}(t) = 1, \quad \forall s' \in \mathcal{S}_f, \ \forall t. \tag{5}$$

On the other hand, if VNF f (or its next VNF) has no instances on server s (or s'), then $X_f^{(s,s')}(t) = 0$ for each time slot t. Note that dynamic request steering can be implemented by adopting VNFs-enabled SDN switches [20]. We denote all chaining decisions by $\boldsymbol{X}(t)$.

For each server s and VNF $f \in \mathcal{F}_s$, we define $Y_f^s(t) \in \mathbb{Z}_+^R$ as the allocated resource vector to f's instance. To ensure any allocation with at least one CPU core and other resources, or without any resources at all, we restrict the choice of $Y_f^s(t)$ to a finite set of options $\mathcal{O}_f$. Note that $\emptyset \in \mathcal{O}_f$ for all f, *i.e.*, the option of no resource allocation is always available. Besides, the total allocated resources should not exceed server s's resource capacity, *i.e.*,

$$\sum_{f \in \mathcal{F}_s} Y_f^s(t) \preceq \boldsymbol{c}_s, \ \ \forall s \in \mathcal{S}, \ \forall t. \tag{6}$$

Note that $Y_f^s(t) = \emptyset$ for all the time if $f \notin \mathcal{F}_s$. Given resource allocation $Y_f^s(t)$, the instance can process and forward at most $\phi_f(Y_f^s(t))$ requests, where $\phi_f(\cdot)$ is assumed to be estimated from system logs. Due to time slot length limit, a VNF instance can't process too many requests. Thus we assume $\phi_f(\cdot) \leq \phi_{max}$ for some constant ϕ_{max}. We denote all allocation decisions by $\boldsymbol{Y}(t)$.

F. System Workflow and Queueing Dynamics

In time slot t, the system workflow proceeds as follows. At the beginning of time slot t, system components (including traffic classifier, VNF instances, and servers) collect all available system dynamics to make request admission, service chaining, and resource allocation decisions $[\boldsymbol{\mu}(t), \boldsymbol{X}(t), \boldsymbol{Y}(t)]$. According to the decisions, traffic classifier admits new requests for different network services. VNF instances steer the requests which are processed in time slot $(t-1)$ to their next VNF's instances. Meanwhile, every server allocates the resources to its resident VNF instances. The instances then process the requests from their respective queues. At the end of time slot t, the prediction window for each network service k moves one slot ahead. Thus given $\mu_k(t)$, prediction queue $Q_k^p(t)$ is updated as follows

$$Q_k^p(t+1) = [Q_k^p(t) - \sum_{s \in \mathcal{S}_f} \mu_k^s(t)]^+ + A_k(t+D+1). \tag{7}$$

With the above workflow, we have the subsequent queueing dynamics for different VNF instances.

Instances of Ingress VNFs: For every network service k and its respective ingress VNF f, there are $\mu_k^s(t)$ admitted requests to f's instance on server $s \in \mathcal{S}_f$. Thus the update function for queue backlog $Q_f^s(t)$ is

$$Q_f^s(t+1) = \left[Q_f^s(t) - \phi_f(Y_f^s(t)) + \mu_{k_f}^s(t)\right]^+. \tag{8}$$

Instances of Non-Ingress VNFs: For the instance of VNF $f \notin \mathcal{F}_{in}$ on server s, if $X_{p(f)}^{(s',s)}(t) = 1$, then the instance will receive processed requests from the instance of VNF $p(f)$ on server s'; otherwise, the instance will receive no new requests. Thus the queueing update function is

$$Q_f^s(t+1) \leq \left[Q_f^s(t) - \phi_f(Y_f^s(t)) + \sum_{s' \in \mathcal{S}_{p(f)}} X_{p(f)}^{(s',s)}(t) \cdot B_{p(f)}^{s'}(t)\right]^+ \tag{9}$$

where $B_{p(f)}^{s'}(t) \triangleq \phi_{p(f)}\left(Y_{p(f)}^{s'}(t-1)\right)$, *i.e.*, the allocated service rate for the instance of $p(f)$ on server s' in time $(t-1)$. The inequality is due to that the actual number of untreated requests may be less than the service rate in time $(t-1)$. All requests processed by the last instances of service chains are considered finished. The vector $[B_f^s(t)]_{s,f}$ is denoted by $\boldsymbol{B}(t)$. Figure 3 shows an instance of our queue model.

G. Optimization Objectives

Communication Cost: Recall that transferring a request over link (s', s) incurs a communication cost $w_{s',s}(t)$, *e.g.*, the number of hops or round-trip times. Low communication costs are highly desirable for responsiveness of requests. In time slot t, given the service chaining decisions, the communication cost between server s and s' is

$$m_{s',s}(t) \triangleq \hat{m}_{s',s}(\boldsymbol{X}(t)) = \sum_{f \in \mathcal{F}_{nt}} B_f^{s'}(t) X_f^{(s',s)}(t) w_{s',s}(t). \tag{10}$$

Thus the total communication cost in time t is

$$m(t) \triangleq \hat{m}\left(\boldsymbol{X}(t)\right) = \sum_{s',s \in \mathcal{S}} \hat{m}_{s',s}(\boldsymbol{X}(t)). \tag{11}$$

Energy Cost: Efficient resource utilization for servers is another important objective to achieve in NFV systems. Given the resource allocation $Y_f^s(t)$, we define the corresponding computational cost in time t as $\boldsymbol{\lambda}^T Y_f^s(t)$, where $\boldsymbol{\lambda} \in \mathbb{Z}_+^R$ is a constant vector, with each entry λ_i as the unit cost of i-th type of server resources. The total energy cost in time t is

$$g(t) \triangleq \hat{g}(\boldsymbol{Y}(t)) = \sum_{s\in\mathcal{S}} \sum_{f\in\mathcal{F}_s} \boldsymbol{\lambda}_s^T Y_f^s(t). \tag{12}$$

Queue Stability: Considering the responsiveness of requests and scarcity of computational resources such as memory and cache, it is also imperative to achieve load balancing in the system, so that no queue backlogs would be overloaded. We denote the weighted total queue backlog size in time t as

$$h(t) \triangleq \hat{h}(\boldsymbol{Q}(t)) = \sum_{k=1}^{K} Q_k^p(t) + \alpha \sum_{s\in\mathcal{S}} \sum_{f\in\mathcal{F}_s} Q_f^s(t) \tag{13}$$

where α is a constant that weights the importance of stabilizing instances' queue backlogs compared to prediction queues. Accordingly, we define the queue stability [21] as

$$\limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\left\{h(t)\right\} < \infty. \tag{14}$$

H. Problem Formulation

Based on the above models, we formulate the following stochastic network optimization problem (**P1**) that aims at the joint minimization of time-average expectations of weighted communication cost and energy cost while ensuring queue stability. With such formulation, we explore the potential trade-off among different system metrics.

$$\begin{aligned} \textbf{P1:} \quad \underset{\{\boldsymbol{\mu}(t),\boldsymbol{X}(t),\boldsymbol{Y}(t)\}_t}{\text{Minimize}} \quad & \limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\left\{m(t)+g(t)\right\} \\ \text{Subject to} \quad & \mu_k^s(t) \in \mathbb{Z}_+, \ \forall k \text{ and } s \in \mathcal{S}_{f_{k,1}} \\ & Y_f^s(t) \in \mathcal{O}_f, \forall s \in \mathcal{S}, f \in \mathcal{F}_s \\ & (2), (5), (6), (14) \end{aligned} \tag{15}$$

where $\gamma \geq 0$ is a constant that weights the relative importance of energy efficiency to reducing communication cost.

IV. Algorithm Design and Performance Analysis

We present POSCARS, an online and predictive algorithm that solves problem **P1** through a series of online decisions, followed by its performance analysis and three variants.

A. Algorithm Design

Problem **P1** is challenging to solve due to time-varying system dynamics, the online nature of request arrivals, and complex interaction between successive VNF instances. Therefore, instead of solving problem **P1** directly, we adopt Lyapunov optimization techniques [21] and the concept of *opportunistically minimizing an expectation* [21] to transform the long-term stochastic optimization problem into a series of sub-problems over time slots. With the non-trivial transformation, we achieve the optimum of **P1** approximately, by solving each of the sub-problems optimally in each time slot with limited instant system dynamics. In each time slot t, we solve

$$\begin{aligned} \textbf{P2:} \quad \underset{\boldsymbol{\mu},\boldsymbol{X},\boldsymbol{Y}}{\text{Minimize}} \quad & \sum_{k=1}^{K} \sum_{s\in\mathcal{S}_{f_{k,1}}} \left[-Q_k^p(t) + \alpha Q_{f_{k,1}}^s(t)\right] \mu_k^s \\ & + \sum_{f\in\mathcal{F}_{nt}} \sum_{s'\in\mathcal{S}_f} \sum_{s\in\mathcal{S}_{n(f)}} l_f^{(s',s)}(t) X_f^{(s',s)} \\ & + \sum_{s\in\mathcal{S}} \sum_{f\in\mathcal{F}_s} r_f^s(t, Y_f^s) \\ \text{Subject to} \quad & (2), (5), (6) \text{ and } \mu_k^s \in \mathbb{Z}_+, \ \forall k, s \in \mathcal{S}_{f_{k,1}} \\ & X_f^{(s',s)} \in \{0,1\}, \ \forall s', s \in \mathcal{S}, \ f \in \mathcal{F}_s \\ & Y_f^s \in \mathcal{O}_f \ \forall s \in \mathcal{S}, f \in \mathcal{F}_s. \end{aligned} \tag{16}$$

where $l_f^{(s',s)}(t)$ are defined as

$$l_f^{(s',s)}(t) \triangleq \left[V w_{s',s}(t) + \alpha Q_{n(f)}^s(t)\right] B_f^{s'}(t), \tag{17}$$

such that V is a positive parameter that weights the importance of minimizing system costs compared to stabilizing system queue backlogs. and $r_f^s(t, Y)$ as

$$r_f^s(t, Y) \triangleq V\gamma \boldsymbol{\lambda}_s^T Y - \alpha Q_f^s(t) \phi_f(Y). \tag{18}$$

In fact, problem **P2** can be further decomposed based on different scheduling decisions, including the sub-problems for request admission, service chaining, and resource allocation. We denote their decisions in one time slot by $\boldsymbol{\mu}$, $\boldsymbol{X}$, and $\boldsymbol{Y}$, respectively. We propose POSCARS and show its pseudocode in Algorithm 1.

Remark 1: POSCARS solves the sub-problems through three sub-routines in a distributed manner (line 2-11 for request admission, line 13-16 for service chaining, and line 19-32 for resource scheduling). In practice, the request admission sub-routine can be implemented on each traffic classifier; meanwhile, one can deploy the service chaining and resource scheduling sub-routines on the hypervisor of each server.

Remark 2: Regarding request admission, when all instances are more loaded than the prediction queue. in order not to overload any instances, POSCARS admits only untreated requests at current time slot and spreads them evenly onto least loaded instances. However, when instances have smaller queue backlogs than the prediction queue, POSCARS admits all future requests and assigns them to the least loaded instances.

Remark 3: POSCARS decides the service chaining by jointly considering instances' queue backlog and the communication cost. Recall the definition in (17), where the weighted summation $\alpha Q_{n(f)}^s(t) + V w_{s',s}(t)$ actually reflects the unit price of sending a request from VNF f's instance on server s' to the instance of its next VNF on server s. If the target instance is heavily loaded, there will be a high price of forwarding the request to that instance. Meanwhile, a large communication cost $w_{s',s}(t)$ will also make it less willing to choose the target instance.

Remark 4: On server s, the resource allocation is decided by jointly considering the resource costs and the queue backlogs of its resident instances. Particularly, we regard the term $V\gamma\boldsymbol{\lambda}_s - \alpha Q_f^s(t)\boldsymbol{\theta}_f$ as the unit net cost vector of resources allocated to the instance of VNF $f \in \mathcal{F}_s$. Regarding the unit net cost of type-i resource, *i.e.*, $V\gamma\lambda_{s,i} - \alpha Q_f^s(t)\theta_{f,i}$, it is the weighted difference between the unit cost $\lambda_{s,i}$ of type-i

Algorithm 1 POSCARS (Predictive Online Service Chaining And Resource Scheduling) in one time slot

1: Initially in time slot t, given backlog sizes $\boldsymbol{Q}(t)$, service rates $\boldsymbol{B}(t)$, energy costs $\{\boldsymbol{\lambda}_s\}$, and communication costs $\boldsymbol{w}(t)$. Output: chaining and scheduling decisions.
2: **for** every network service $k \in \{1, 2, \ldots, K\}$
3: %% Request admission for ingress VNF $f_{k,1}$
4: The traffic classifier first finds the set $S^*_{f_{k,1}}$ of servers that host the least loaded instances of VNF $f_{k,1}$.
5: **if** $\alpha Q^s_{f_{k,1}}(t) > Q^p_k(t)$ for all $s \in S^*_{f_{k,1}}$ **then**
6: Admit the $Q^{(0)}_k(t)$ untreated requests at current time.
7: **else**
8: Admit all $Q^p_k(t)$ untreated requests.
9: **endif**
10: Spread admitted request evenly to least loaded instances.
11: **endfor**
12: %% Service chaining
13: **for** every non-terminal VNF $f \in \mathcal{F}_{nt}$:
14: **for** the instance of f on server $s' \in \mathcal{S}_f$:
15: Forward its processed requests to one of the servers from $\mathcal{S}_{n(f)}$ with minimum $l^{(s',s)}_f(t)$.
16: **endfor**
17: **endfor**
18: %% Resource scheduling
19: **for** every server $s \in \mathcal{S}$:
20: Initialize an empty lookup table $\mathcal{Y}_{cand}$ and set $\mathcal{F}_{alloc}$.
21: Set $\mathcal{Y}_{cand}[r^s_f(t, Y)] \leftarrow (f, Y),\ \forall\ f \in \mathcal{F}_s$ and $Y \in \mathcal{O}_f$
22: **while** $|\mathcal{Y}_{cand}| > 0$:
23: Choose the minimum r^* among all keys of $\mathcal{Y}_{cand}$.
24: Select its associated f^* and Y^*.
25: Remove entry with key r^* from $\mathcal{Y}_{cand}$.
26: **if** $r^* < 0$ and $\sum_{f \in \mathcal{F}_{alloc}} Y^s_f(t) + Y^* \preceq \boldsymbol{c}_s$:
27: Allocate resource to f^* according to Y^*.
28: $\mathcal{F}_{alloc} \leftarrow \mathcal{F}_{alloc} + \{f^*\}$.
29: Remove all entries related to f^*.
30: **endif**
31: **endwhile**
32: **endfor**

resource and the queue backlog $Q^s_f(t)$ of the instance. A high unit resource cost will result in a prudent allocation. On the other hand, a sufficiently large queue backlog will make the allocation more worthwhile. In both cases, POSCARS selects the set of resource allocation decisions $\{Y^s_f\}_{f \in \mathcal{F}^s}$ that satisfy constraint (6) and minimize the total net cost.

B. Performance Analysis

We present the complexity analysis of POSCARS in one time slot as follows. Recall that POSCARS can be run in a distributed manner. Thus each network service takes $O(|\mathcal{S}|)$ time to make request admission decisions (line 4-10). Next, each non-terminal VNF instance selects and forwards requests to its successors in $O(|\mathcal{S}|)$ time (line 15). Every server takes $O(|\mathcal{F}|)$ time to initialize the lookup table (line 21-23) and $O(\Omega_{max} \times |\mathcal{F}|)$ time to decide the resource allocation, where Ω_{max} is the maximum number of applicable resource allocation for any VNF instance. Each of the decision making takes only linear-time complexity in every time slot.

On the other hand, without predictive scheduling, we show that POSCARS achieves an $[O(V), O(1/V)]$ trade-off between the time-averages of total queue backlog size and total cost via a tunable parameter V. Particularly, setting a large value for parameter V incentivizes POSCARS to steer request traffic from VNF instances to their successive instances in nearby servers. In such a way, communication cost can be effectively reduced; however, some servers may become hot spots and hence the total queue backlog size will increase as well. Conversely, a smaller value of parameter V leads to a more balanced queue backlogs across servers, at the cost of growing communication cost. We delegate the detailed proof to Appendix-B. With predictive scheduling, POSCARS leads to even better performance, as will be found in simulation results.

C. Practical Issues and Variants of POSCARS

The distributed nature of POSCARS requires each VNF instance to gather relevant system dynamics on its own. However, the probing process may incur considerable sampling overheads and additional latencies. Meanwhile, each instance makes its independent decision based on the sampled information at the beginning of a time slot. Therefore, instances may blindly choose the same lowest-cost instance, without knowing others' choices. The chosen instance will then become overloaded due to the non-coordinated decisions. An alternative is to perform sampling before sending each request. Nonetheless, this method suffers from the messaging overheads of frequent samplings. A possible compromise is to split the processed requests into batches, then sample and schedule for each batch separately.

To mitigate such issues, we propose the following variants of POSCARS, by adopting the ideas from recent randomized load balancing techniques, such as **The-Power-of-d-Choices** [22], **Batch-Sampling** [23], and **Batch-Filling** [24].

POSCARS with The-Power-of-d-Choices (P-Pod): To reduce sampling overheads, we apply the idea of *The-Power-of-d-Choices* to POSCARS. Particularly, every non-terminal instance probes only the d instances uniformly randomly from its next VNF. Next, the instance chooses to send all its processed requests to the lowest-cost instance among the d samples. In such a way, each instance requires only few times of sampling to decide its target instance. Although the selected instance may not be the least-cost one, our later simulation results show that the reduced sampling brings only a mild increase in the total cost.

The above variant significantly reduces the sampling overheads. However, the issue of non-coordinated decision making still remains. To mitigate the issue, we apply *batch-sampling* [23] and *batch-filling* [24] techniques to forward each request batch, by splitting instances processed requests into batches, each with a size of b. When $b = 1$, we actually perform scheduling for each request separately. When b is greater than the number of processed requests, then scheduling is only performed once in a time slot, degenerating to POSCARS. We propose another two variants of POSCARS as follows.

POSCARS with Batch-Sampling (P-BS): Given an instance with z batches of requests, it probes $d_{\text{bs}}z$ instances uniformly randomly from its next VNF, where d_{bs} is the respective probe ratio. Then the instance sends the z request batch to the least-cost z instances, with each batch to a distinct target instance.

POSCARS with Batch-Filling (P-BF): Given an instance with z request batches, it probes $d_{\text{bf}}z$ instances uniformly randomly from its next VNF. Then it forwards the request batches one by one. Each batch is sent to the least-cost instance among the $d_{\text{bf}}z$ samples. The chosen instance's cost is updated after it receives the batch of requests.

V. Simulation

We conduct trace-driven simulations to evaluate the performance of POSCARS and its variants. The request arrival measurements are drawn from real-world systems [25], with a mean arrival rate of 25.5 per time slot (10ms) and mean inter-arrival time of 0.594ms. Besides, we conduct simulations with the Poisson request arrivals with the same rate of 25.5. All the results are obtained by averaging measurements collected from 50 repeated and independent simulations.

A. Simulation Settings

Substrate Network Topology: We construct the substrate network based on two widely adopted topologies, *i.e.*, Jellyfish [26] and Fat-Tree [27]. Both topologies have a comparable scale to clusters in data center networks, each equipped with 720 switches, 24 servers with deployed VNFs, and the rest 3456 servers as hosts that generate service requests. Particularly, in Fat-Tree, there are 24 pods, each pod containing to 144 servers; amongst them, we choose one server uniformly at random as the one with deployed VNFs and the rest as hosts. Requests can be processed on servers in any pod with the VNF they demand. Between any two servers, request traffic traverses over the shortest path with link capacity of 40Gbps. For each pair of servers, the communication cost per request is proportional to the number of hops of the shortest path between them, with 10% variation.

Server Resources: We consider CPU cores as the resources on each server, since CPUs have become the major bottleneck for request processing in NFV systems [28]–[30]. Servers are heterogeneous, each with a number of CPU cores ranging from 16 to 64. In every time slot, we calculate the power consumption in the unit of utilized CPU cores, with $\lambda_s \in [1, 3]$. Regarding parameter γ, setting it with a greater value would encourage each server to assign most resources to heavily loaded VNF instances. Conversely, a smaller value of γ would lead to more balanced resource allocation among such instances; consequently, this will minimize the impact of imbalanced queue backlogs on the decision making for service chaining. The value setting depends on the objectives to fulfill in real systems. In our simulation, by fixing $\gamma = 1$, we assume that communication cost reduction and system energy efficiency are equally important.

Service Function Chains: We deploy five network services, each with a service chain length varying from 3 to 5. Each service contains at least one of the most commonly-deployed VNFs; *e.g.*, Intrusion Detection System (IDS), Firewall (FW), Load Balancer (LB). The rest VNFs of each service are chosen uniformly from other 30 commonly-used VNFs [31] at random without replacement. For each VNF, the total number of instances ranges from 12 to 18.

Prediction Settings: Network services' traffic often varies in predictability. We denote the average window size by D, and set each service window size by sampling uniformly from $[0, 2\times D]$ at random. We evaluate the cases with perfect and imperfect prediction. For perfect prediction, future request arrivals in the time window are assumed perfectly known to the system and can be pre-served. In practice, such an assumption is not feasible for stateful requests; nonetheless, that can be seen as the extended case of our results with more constraints on request processing. For imperfect prediction, the failure of prediction generally falls into two categories. One is *true-negative* detection, *i.e.*, a request is not predicted to arrive, and as a result, it receives no pre-service before its arrival. The other is *false-positive* detection, *i.e.*, a request that does not exist is predicted to arrive. In this case, the system pre-allocates resources to pre-serve such requests. We consider two extreme cases: one is that we fail to predict the arrivals of all future requests; the other is that we correctly predict the actual future arrivals, and furthermore, some extra arrivals are falsely alarmed. Note that any form of mis-prediction can be seen as a superposition of such two extremes. In addition, we also implement five schemes that forecast request arrivals in the next time slot (with window size $D = 1$), including: 1) Kalman filter (Kalman) [32]; 2) distribution estimator (Distr), which generates the next estimate by independent sampling from the distribution of arrivals learned from historical data; 3) Prophet (FB) [33], Facebook's time-series forecasting procedure; 4) moving average (MA) and 5) exponentially weighted moving average (EWMA) [34].

Baseline Schemes: We compare POSCARS with three baseline schemes: Random, JSQ (Join-the-Shortest-Queue), and OneHop-SCH (OneHop scheduling) [35]. These schemes differ in the service chaining strategy from POSCARS. In Random scheme, each instance uniformly randomly dispatches request to one of its successors. In JSQ scheme, each instance sends its requests to its least-loaded successor. In OneHop-SCH scheme, each instances sends its requests to the successor with least communication costs and idle capacities.

Variants of POSCARS: To compare the performance of POSCARS and its variants, we evaluate them under different settings. For each of the variants, we vary their probe ratio (d for P-Pod, d_{bs} for P-BS, and d_{bf} for P-BF) from 2 to 5, and fix the batch size for P-BS and P-BF as 5 requests per batch. We omit the cases when the ratio is 1 and greater than 5. Notice that the former corresponds to the random scheme and actually leverages no load information; the latter leads to excessively fined-grained control since it induces too much sampling overheads.

Request Response Time Metric: To evaluate the impact of predictive scheduling, we define a request's response time as the number of time slots from its actual arrival to its eventual completion. Thus if a request is pre-served before it arrives, then the system is assumed to respond to the request upon its arrival, and the request will experience a zero response time.

B. Performance Evaluation under Perfect Prediction

Intuitively, POSCARS is promising to shorten the requests' response time by exploiting predicted information and pre-allocating idle system resources to pre-serve future requests. Thus the essential benefits of predictive scheduling comes from the load balancing in the temporal dimension. To verify such intuition, we first consider the case with perfectly predicted arrivals, and evaluate POSCARS with ($D > 0$) and without ($D = 0$) prediction, against the baseline schemes.

Average response time vs. window size D: Figure 4 shows the performance of the different schemes under Jellyfish and Fat-Tree topology. The response times induced

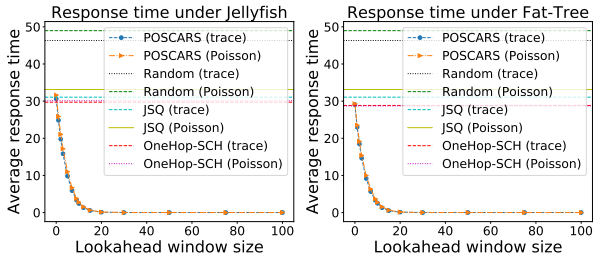


Fig. 4: Average response time (ms) with various window sizes given trace and Poisson arrival process, under different topologies.

by the baseline schemes remain constant since they do not involve predictive scheduling. Random incurs the highest response time ($\sim$ 47ms), since it disregards information about workloads or communication cost when dispatching requests. JSQ does much better ($\sim$ 32ms) because requests are always greedily forwarded to the least-loaded successors. OneHop-SCH outperforms the previous two by jointly taking the workloads and communication cost into consideration. Meanwhile, without prediction ($D = 0$), POSCARS achieves comparable performance with OneHop-SCH; but as D increases from 0 to 20, we observe a significant reduction in the average response time under both topologies; *e.g.*, from 29.1ms to 0.5ms under Fat-Tree topology. The marginal reduction diminishes as D further increases, and eventually, remains at around 0.2ms.

Insight: In practice, due to traffic variability, it is often not realistic to achieve high predictability (large D). However, the results show that, only mild-value of future information suffices to POSCARS's shortening requests response time effectively and achieving load-balancing in the temporal dimension. With more future information, the reduction diminishes since the idle system resources have already been depleted.

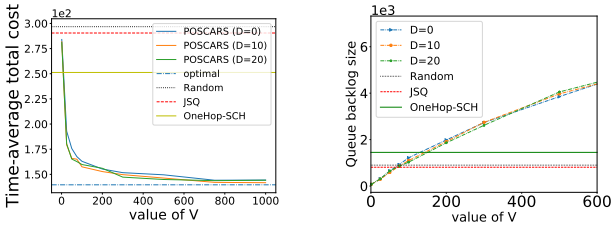
Considering the qualitative similarities among curves with different settings, we present the results under Fat-Tree topology and trace-driven request loads particularly.

Backlog-cost trade-off with parameter V: Recall from Section III.B that the value of parameter V controls the backlog-cost trade-off. Figure 5 (a) and (b) verify such a trade-off. Figure 5 (a) compares the time-average communication costs of POSCARS with $D = 0$, 10, 20, against baselines. Both Random and JSQ induce high total cost since their decision making disregards the resultant communication costs and the heterogeneity of servers in terms of energy costs. OneHop-SCH further lowers the total cost by about 13.6%, by taking its advantages of jointly optimizing cost and queue backlogs based on flow-level statistics. In comparison, given different choices of D, POSCARS achieves close-to-optimal time-average total costs as the value of V rises up to 10^3. Notably, POSCARS excels OneHop-SCH whenever $V > 10$.

However, to reduce system cost, large values of V also lead to increased backlogs. By *Little's theorem* [36], this would increase response time as well. In Figure 5 (b), we see that the total queue backlog size is almost proportional to value of V, exceeding all other baselines as $V > 150$.

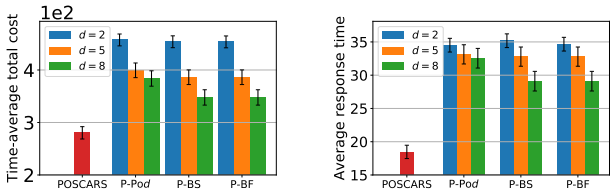
Insight: POSCARS achieves a backlog-cost trade-off with different values of parameter V. By choosing an appropriate value of V from $[10, 150]$, it outperforms the baseline schemes with both lower system costs and smaller queue backlog sizes.

POSCARS and its variants: Upon forwarding requests, POSCARS requires each instance to collect statistics from



(a) Total cost with parameter V (b) Queue size with parameter V

Fig. 5: Queue backlog size under different window sizes.



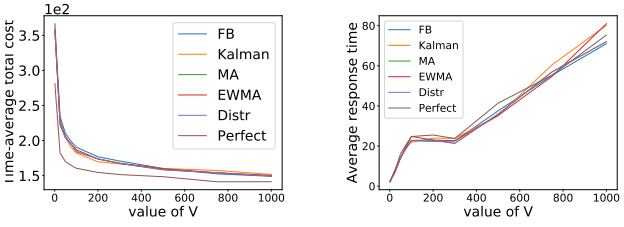
(a) Time-average Total Cost (b) Average Response Time

Fig. 6: Comparison among POSCARS and its variants

all its successors. In practice, this may require non-negligible sampling overheads in face of a large number of instances. In Section III.C, we propose three variants of POSCARS, *i.e.*, P-Pod, P-BS, and P-BF. These variants trade off optimality of decision making for reduction in sampling overheads and complexity [24] from $O(n)$ to $O(1)$, where n denotes the total number of candidate instances. Figure 6 evaluates the total costs and average response time induced by POSCARS and its variants, with parameter $V = 10$, $D = 1$, batch size of 5 for P-BS and P-BF, and the probe ratio $d = d_{\text{bs}} = d_{\text{bf}} \in \{2, 5, 8\}$.

In Figure 6 (a), we see that POSCARS achieves the lowest total cost, since each instance's decision making is based on the full dynamics of its succeeding instances. For each variant, we see a cut-down in the total cost by up to 22.1% as d increases from 2 to 8. Similarly, from Figure 6 (b), we also observe a reduction in response time from about 34.3ms by up to 17.6%. Among the three variants, P-BS and P-BF induce more reduction in both costs and response time than P-Pod, because aggregated sampling is often more conducive to lowering the cost [23].

Insight: By sampling partial system dynamics for decision making, variants of POSCARS trade off optimality for reduction in sampling overheads and complexity. Owing to aggregated sampling, P-BF and P-BS outperforms P-Pod in terms of both lower total cost and response time.



(a) Time-average total cost (b) Average response time

Fig. 7: Performance of prediction schemes with $D = 1$ and $\alpha = 10$.

C. Performance Evaluation under Imperfect Prediction

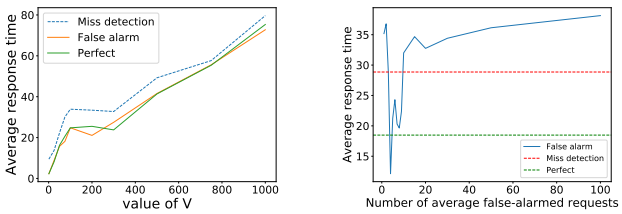
In practice, prediction errors are inevitable due to dataset bias and noise. To explore the fundamental limits of predictive scheduling, we evaluate the impact of imperfect prediction on the system performance.

Total cost and response time vs. V: Figure 7 compares the time-average total cost and average response time induced by different forecasting schemes and perfect scheduling using POSCARS. In Figure 7 (a), we observe that all forecasting schemes incur higher time-average total costs than predictive scheduling by up to 35.2%. The reason is as follows. Recall that the prediction under these forecasting schemes are imperfect, with both true-negative and false-positive detection. Particularly, the system pre-allocates extra resources to preserve false-positive requests, resulting in higher total costs. Figure 7 (b), shows the overall ascending trend proportional to increased V. This is due to that larger values of V lead to larger queue backlog sizes, and by *Little's theorem* [36], larger queue backlog sizes implies longer response time. However, we also see that, even under imperfect prediction, predictive scheduling does not necessarily lead to longer response time than that under perfect prediction.

To figure out the reason, we consider two extreme cases. One is all-*true-negative*, *i.e.*, during each time slot, all future request arrivals in the lookahead window are true-negative. Notice that this case is equivalent to the case without predictive scheduling ($D = 0$), since no requests will be pre-allocated resources. The other is all-*false-positive*, *i.e.*, all future request arrivals are perfectly predicted, and besides, some extra requests are wrongly predicted to arrive.

Perfect prediction with two extremes: Figure 8 (a) compares average response times under perfect prediction and the two extremes, with D=5, α=10, and 5 false-positive requests on average. Overall, the average response time is proportional to the value of V. Miss detection incurs higher response time than the other two, because it does not pre-serve any requests before they arrive. On the other hand, perfect prediction and false alarm do not necessarily outperform the other with lower response times. This is because of two consequences of false alarm. The *first* is that false-positive requests will consume extra system resources and prolong request queue backlogs, thus leading to longer response times. The *second* is that, according to line 5 - 9 in Algorithm 1, false-positive requests result in a greater prediction queue backlog size. That forces POSCARS to admit future requests more frequently, thus conducing to shorter response times. The same effect can be achieved by tuning the parameter α – greater values of α lead to less frequent admission.

How do these two consequences interplay? The question is answered by Figure 8 (b), where the number of average false-positive requests varies from 0 to 100, with $D = 5$, $\alpha = 10$, and $V = 50$. When the average number of false-positive requests increases from 0 to 5, the resultant response time falls even lower than that under perfect prediction. In such cases, the second consequence dominates – mild false alarm leads to more frequent admission, making POSCARS spread requests more evenly among instances. However, as false alarm continues aggravating, the reduction diminishes and the response time grows constantly. In such cases, though the admission frequency is intensified, too much false alarm severely enlarges queue backlog sizes, offsetting and eventually outweighing the effect of load balancing.



(a) Different values of V (b) Different prediction accuracies

Fig. 8: Average response time under three different prediction cases

Insight: Imperfect prediction does not necessarily degrade system performance, such as longer response times. Instead, mild false alarm allows the system to make better use of idle system resources, further shortening response time.

VI. Related Work

Regarding service chaining and resource scheduling problem, existing solutions generally fall into two categories.

Of the first category are the schemes that perform service chaining and resource scheduling in an offline fashion. Typically, they assume the full availability of information about all service requests or flows. Based on flow abstraction, Zhang *et al.* [14] consider the joint optimization for VNF placement and service chaining. They formulate the problem as an ILP problem and develop an efficient rounding-based approximation algorithm with performance guarantee. Yoon *et al.* [37] adopt the BCMP queueing model for VNF service chains and propose heuristics to approximately minimize the expected waiting time of service chains. Wang *et al.* [35] consider the joint optimization of service chaining and resource allocation and develop a greedy scheme that aims to place instances and schedule traffic with minimum link costs, CAPEX, and OPEX. Later, D'Oro *et al.* [38] study service chaining problem from the perspective of congestion games. By formulating the problem as an atomic weighted congestion game, they propose a distributed algorithm that provably converges to the Nash equilibrium. On the other hand, Zhang *et al.* [15] formulate a request-level optimization problem based on steady-state metrics and propose a heuristic scheme by applying techniques from open Jackson queueing network. However, there is no empirical evidence to show that service request arrivals follow Poisson process in NFV systems. Different from existing works, our model and problem formulation assumes no prior knowledge about underlying request traffic. Moreover, instead of offline or even centralized decision making, our solution is capable to perform near-optimal service chaining and scheduling in a computationally efficient, online, and decentralized manner.

Of the second category are the online schemes that process requests upon their arrivals. Under this setting, Mohammadkhan *et al.* [9] formulate the VNF placement for service chaining as a MILP problem based on flow abstraction and develop a heuristic to solve the problem incrementally. Lukovszki *et al.* [39] develop an online algorithm that performs request admission and service chaining with a logarithmic competitive ratio. Zhang *et al.* in [40] propose a novel VNF brokerage service model and online algorithms to predict traffic demands, purchase VMs and deploy VNFs. Further, Fei *et al.* [41] develop an effective algorithm that performs online VNF scheduling and flow routing with predicted flow demand, so as to minimize the impact of inaccurate prediction and the cost

of over-provisioned resources. These schemes either resort to flow-level system dynamics and predicted information for decision making, or perform finer-grained control at the request level to optimize dedicated objectives. Besides, the fundamental benefits and limits of predictive scheduling still remain unexplored. Our model considers those trade-offs and separates the granularity of system state and decision making while exploiting the power of predictive scheduling.

VII. Conclusion

In this paper, we studied the problem of dynamic service chaining and resource scheduling and systematically investigated the benefits of predictive scheduling in NFV systems. We developed a novel queue model that accurately characterizes the system dynamics. Then we formulated a stochastic network optimization problem and then proposed POSCARS, an efficient and decentralized algorithm that performs service chaining and scheduling through a series of online and predictive decisions. Theoretical analysis and trace-driven simulations showed the effectiveness and robustness of POSCARS and its variants in achieving near-optimal system costs while effectively shortening average response time.

Appendix-A
Problem Transformation

To solve problem (15), we adopt the Lyapunov optimization technique [21]. We define the quadratic Lyapunov function as

$$L(\boldsymbol{Q}(t)) \triangleq \frac{1}{2}\left[\sum_{k=1}^{K}\left(Q_k^p(t)\right)^2+\alpha\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}_s}\left(Q_f^s(t)\right)^2\right] \tag{19}$$

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

$$\Delta(\boldsymbol{Q}(t)) \triangleq \mathbb{E}\left\{L(\boldsymbol{Q}(t+1))-L(\boldsymbol{Q}(t))\middle|\boldsymbol{Q}(t)\right\} \tag{20}$$

which measures the conditional expected successive change in queues' congestion state. To avoid overloading any queue backlogs in the system, it is desirable to make the difference as low as possible. However, striving for small queue backlogs may incur considerable communication cost and computation cost. Hence, we should jointly consider both queueing stability and the consequent system costs. Thus we define the drift-plus-penalty function as

$$\begin{aligned}\Delta_V(\boldsymbol{Q}(t)) \triangleq\ & \mathbb{E}\left\{L(\boldsymbol{Q}(t+1))-L(\boldsymbol{Q}(t))\middle|\boldsymbol{Q}(t)\right\}\\ &+V\mathbb{E}\left\{m(t)+\gamma g(t)\middle|\boldsymbol{Q}(t)\right\}\end{aligned} \tag{21}$$

where V is a positive constant that determines the balance between queueing stability and minimizing total system costs.

Next, we show how problem (15) is transformed into (44) in detail. According to (21), the drift-plus-penalty term is

$$\begin{aligned}&\Delta_V(\boldsymbol{Q}(t))\\ =\ &\mathbb{E}\left\{L(\boldsymbol{Q}(t+1))-L(\boldsymbol{Q}(t))+V[m(t)+\gamma g(t)]\middle|\boldsymbol{Q}(t)\right\}.\end{aligned} \tag{22}$$

Remind the definitions of quadratic Lyapunov function in (19), we expand $\Delta_V(\boldsymbol{Q}(t))$

$$\begin{aligned}&\Delta_V(\boldsymbol{Q}(t))\\ &=\frac{1}{2}\mathbb{E}\left\{\sum_{k=1}^{K}\left[Q_k^p(t+1)\right]^2-\sum_{k=1}^{K}\left[Q_k^p(t)\right]^2\middle|\boldsymbol{Q}(t)\right\}\\ &+\frac{\alpha}{2}\sum_{s\in\mathcal{S}}\mathbb{E}\left\{\sum_{f\in\mathcal{F}_s\bigcap\mathcal{F}_{in}}\left[\left(Q_f^s(t+1)\right)^2-\left(Q_f^s(t)\right)^2\right]\middle|\boldsymbol{Q}(t)\right\}\\ &+\frac{\alpha}{2}\sum_{s\in\mathcal{S}}\mathbb{E}\left\{\sum_{f\in\mathcal{F}_s\setminus\mathcal{F}_{in}}\left[\left(Q_f^s(t+1)\right)^2-\left(Q_f^s(t)\right)^2\right]\middle|\boldsymbol{Q}(t)\right\}\\ &+\mathbb{E}\left\{Vm(t)+V\gamma g(t)\middle|\boldsymbol{Q}(t)\right\}.\end{aligned} \tag{23}$$

To derive the upper bound of (23), we show the following inequalities for any non-negative numbers a, b and c as follows

$$\begin{aligned}\left[(a+b-c)^+\right]^2 &\le (a+b-c)^2\\ &=a^2+b^2+c^2+2a(b-c)-2bc\\ &\le a^2+b^2+c^2+2a(b-c)\end{aligned} \tag{24}$$

$$\begin{aligned}\left[(a-c)^++b\right]^2 &\le (a-c)^2+b^2+2b(a-c)^+\\ &=a^2+c^2-2ac+b^2+2b(a-c)^+\\ &\le a^2+c^2-2ac+b^2+2ba\\ &=a^2+b^2+c^2+2a(b-c)\end{aligned} \tag{25}$$

where the last inequality in (25) holds since for $a, c \ge 0$,

$$a\ge 0 \text{ and } a\ge a-c \tag{26}$$

and thus $a \ge \max\{a-c,0\} = [a-c]^+$. Thereby, with the queueing update equations (7)-(9), we obtain

1) for $k \in \{1,\dots,K\}$,

$$\begin{aligned}\left[Q_k^p(t+1)\right]^2 &= \left\{\left[Q_k^p(t)-\mu_k(t)\right]^+ + A_k(t+D_k+1)\right\}^2\\ &\le \left[Q_k^p(t)\right]^2+\left[A_k(t+D_k+1)\right]^2+\left[\mu_k(t)\right]^2\\ &\quad +2Q_k^p(t)\left[A_k(t+D_k+1)-\mu_k(t)\right].\end{aligned} \tag{27}$$

2) for $s\in\mathcal{S}$ and $f\in\mathcal{F}_s\bigcap\mathcal{F}_{in}$,

$$\begin{aligned}\left[Q_f^s(t+1)\right]^2 &= \left\{\left[Q_f^s(t)-\phi_f(Y_f^s(t))+\mu_{k_f}^s(t)\right]^+\right\}^2\\ &\le \left[Q_f^s(t)\right]^2+\left[\mu_{k_f}^s(t)\right]^2+\left[\phi_f(Y_f^s(t))\right]^2\\ &\quad +2Q_f^s(t)\left[\mu_{k_f}^s(t)-\phi_f(Y_f^s(t))\right].\end{aligned} \tag{28}$$

3) for $s\in\mathcal{S}$ and $f\in\mathcal{F}_s\setminus\mathcal{F}_{in}$,

$$\begin{aligned}&\left[Q_f^s(t+1)\right]^2\\ &=\left\{\left[Q_f^s(t)-\phi_f(Y_f^s(t))+\sum_{s'\in\mathcal{S}_{p(f)}}X_{p(f)}^{(s',s)}(t)\cdot B_{p(f)}^{s'}(t)\right]^+\right\}^2\end{aligned}$$

$$\leq \left[Q_f^s(t)\right]^2 + \left[\sum_{s'\in\mathcal{S}_{p(f)}} X_{p(f)}^{(s',s)}(t)B_{p(f)}^{s'}(t)\right]^2 + \left[\phi_f(Y_f^s(t))\right]^2 + 2Q_f^s(t)\left[\sum_{s'\in\mathcal{S}_{p(f)}} X_{p(f)}^{(s',s)}(t)B_{p(f)}^{s'}(t) - \phi_f(Y_f^s(t))\right]. \tag{29}$$

By injecting (27) – (29) into (23), we obtain

$$\begin{aligned}
&\Delta_V(\boldsymbol{Q}(t)) \\
&\leq \frac{1}{2}\mathbb{E}\left\{\sum_{k=1}^{K}\left[A_k(t+D_k+1)\right]^2 + \left[\mu_k(t)\right]^2 \middle| \boldsymbol{Q}(t)\right\} \\
&+ \frac{\alpha}{2}\mathbb{E}\left\{\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}_s\bigcap\mathcal{F}_{in}}\left[\left[\mu_{k_f}^s(t)\right]^2 + \left[\phi_f(Y_f^s(t))\right]^2\right] \middle| \boldsymbol{Q}(t)\right\} \\
&+ \frac{\alpha}{2}\mathbb{E}\left\{\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}_s\setminus\mathcal{F}_{in}}\left[\left[\sum_{s'\in\mathcal{S}_{p(f)}} X_{p(f)}^{(s',s)}(t)B_{p(f)}^{s'}(t)\right]^2 + \left[\phi_f(Y_f^s(t))\right]^2\right] \middle| \boldsymbol{Q}(t)\right\} \\
&+ \mathbb{E}\left\{\sum_{k=1}^{K} Q_k^p(t)\left[A_k(t+D_k+1) - \mu_k(t)\right] \middle| \boldsymbol{Q}(t)\right\} \\
&+ \alpha\mathbb{E}\left\{\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}_s\bigcap\mathcal{F}_{in}} Q_f^s(t)\left[\mu_{k_f}^s(t) - \phi_f(Y_f^s(t))\right] \middle| \boldsymbol{Q}(t)\right\} \\
&+ \alpha\mathbb{E}\left\{\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}_s\setminus\mathcal{F}_{in}} Q_f^s(t)\left[\sum_{s'\in\mathcal{S}_{p(f)}} X_{p(f)}^{(s',s)}(t)B_{p(f)}^{s'}(t) - \phi_f(Y_f^s(t))\right] \middle| \boldsymbol{Q}(t)\right\} + \mathbb{E}\left\{Vm(t) + V\gamma g(t) \middle| \boldsymbol{Q}(t)\right\}
\end{aligned} \tag{30}$$

Remind the boundedness of all the request arrivals, service capacities on switches and controllers, and according to (2), for $k \in \{1, \ldots, K\}$, we have

$$\mu_k^s(t) \leq \mu_k(t) \leq Q_k^p(t) = \sum_{d=0}^{D_k} Q_k^{(d)}(t) \leq (D+1)\cdot a_{max} \tag{31}$$

and

$$\begin{aligned}
&\Delta_V(\boldsymbol{Q}(t)) \\
&\leq \frac{1}{2}\left[K(a_{max})^2 + K(D+1)^2(a_{max})^2\right] \\
&\quad + \frac{\alpha}{2}Kb_{max}\left[(D+1)^2(a_{max})^2 + (\phi_{max})^2\right] \\
&\quad + \frac{\alpha}{2}Kb_{max}\left[(b_{max})^2(\phi_{max})^2 + (\phi_{max})^2\right] \\
&\quad + \mathbb{E}\left\{\sum_{k=1}^{K} Q_k^p(t)\left[A_k(t+D_k+1) - \mu_k(t)\right] \middle| \boldsymbol{Q}(t)\right\} \\
&\quad + \mathbb{E}\left\{\sum_{k=1}^{K} Q_k^p(t)\left[A_k(t+D_k+1) - \mu_k(t)\right] \middle| \boldsymbol{Q}(t)\right\} \\
&\quad + \alpha\mathbb{E}\left\{\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}_s\bigcap\mathcal{F}_{in}} Q_f^s(t)\left[\mu_{k_f}^s(t) - \phi_f(Y_f^s(t))\right] \middle| \boldsymbol{Q}(t)\right\} \\
&\quad + \alpha\mathbb{E}\left\{\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}_s\setminus\mathcal{F}_{in}} Q_f^s(t)\left[\sum_{s'\in\mathcal{S}_{p(f)}} X_{p(f)}^{(s',s)}(t)B_{p(f)}^{s'}(t) - \phi_f(Y_f^s(t))\right] \middle| \boldsymbol{Q}(t)\right\} + \mathbb{E}\left\{Vm(t) + V\gamma g(t) \middle| \boldsymbol{Q}(t)\right\}
\end{aligned} \tag{32}$$

where $b_{max} \triangleq \max_{f\in\mathcal{F}}|\mathcal{S}_f|$ is the maximum number of instances among all VNFs. Next, we define B as

$$\begin{aligned}
B \triangleq\ & \frac{1}{2}\left[K(a_{max})^2 + K(D+1)^2(a_{max})^2\right] \\
& + \frac{\alpha}{2}Kb_{max}\left[(D+1)^2(a_{max})^2 + (\phi_{max})^2\right] \\
& + \frac{\alpha}{2}Kb_{max}\left[(b_{max})^2(\phi_{max})^2 + (\phi_{max})^2\right]
\end{aligned} \tag{33}$$

Substituting (33) into (32), and canceling the terms that are irrelevant to the decision variables $\boldsymbol{\mu}(t)$, $\boldsymbol{X}(t)$, and $\boldsymbol{Y}(t)$, we define $C(\boldsymbol{Q}(t))$ as

$$C(\boldsymbol{Q}(t)) = \mathbb{E}\left\{\sum_{k=1}^{K} Q_k^p(t)A_k(t+D_k+1) \middle| \boldsymbol{Q}(t)\right\} \tag{34}$$

Next, we obtain

$$\begin{aligned}
&\Delta_V(\boldsymbol{Q}(t)) \\
&\leq B + C(\boldsymbol{Q}(t)) - \mathbb{E}\left\{\sum_{k=1}^{K} Q_k^p(t)\mu_k(t) \middle| \boldsymbol{Q}(t)\right\} \\
&+\alpha\mathbb{E}\left\{\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}_s\bigcap\mathcal{F}_{in}} Q_f^s(t)\left[\mu_{k_f}^s(t) - \phi_f(Y_f^s(t))\right] \middle| \boldsymbol{Q}(t)\right\} \\
&+\alpha\mathbb{E}\left\{\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}_s\setminus\mathcal{F}_{in}} Q_f^s(t)\left[\sum_{s'\in\mathcal{S}_{p(f)}} X_{p(f)}^{(s',s)}(t)B_{p(f)}^{s'}(t) - \phi_f(Y_f^s(t))\right] \middle| \boldsymbol{Q}(t)\right\} + V\mathbb{E}\left\{m(t) + \gamma g(t) \middle| \boldsymbol{Q}(t)\right\}
\end{aligned} \tag{35}$$

Next, remind the definition of total communication cost and total computation cost in (11) and (12), we have

$$\begin{aligned}
&\Delta_V(\boldsymbol{Q}(t)) \leq B + C(\boldsymbol{Q}(t)) - \mathbb{E}\left\{\sum_{k=1}^{K} Q_k^p(t)\mu_k(t) \middle| \boldsymbol{Q}(t)\right\} \\
&+\alpha\mathbb{E}\left\{\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}_s\bigcap\mathcal{F}_{in}} Q_f^s(t)\left[\mu_{k_f}^s(t) - \phi_f(Y_f^s(t))\right] \middle| \boldsymbol{Q}(t)\right\} \\
&+\alpha\mathbb{E}\left\{\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}_s\setminus\mathcal{F}_{in}} Q_f^s(t)\left[\sum_{s'\in\mathcal{S}_{p(f)}} X_{p(f)}^{(s',s)}(t)B_{p(f)}^{s'}(t) - \phi_f(Y_f^s(t))\right] \middle| \boldsymbol{Q}(t)\right\} + V\gamma\mathbb{E}\left\{\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}_s} \boldsymbol{\lambda}_s^T Y_f^s(t) \middle| \boldsymbol{Q}(t)\right\} \\
&+ V\mathbb{E}\left\{\sum_{(s',s)\in\mathcal{E}}\sum_{f\in\mathcal{F}_{nt}} B_f^{s'}(t)X_f^{(s',s)}(t)w_{s',s}(t) \middle| \boldsymbol{Q}(t)\right\}
\end{aligned} \tag{36}$$

Note that

$$\begin{aligned}
&\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}_s\bigcap\mathcal{F}_{in}} Q_f^s(t)\mu_{k_f}^s(t)\\
=&\sum_{f\in\mathcal{F}_{in}}\sum_{s\in\mathcal{S}_f} Q_f^s(t)\mu_{k_f}^s(t)=\sum_{k=1}^{K}\sum_{s\in\mathcal{S}_{f_{k,1}}} Q_{f_{k,1}}^s(t)\mu_k^s(t)\\
=&\sum_{k=1}^{K}\sum_{s\in\mathcal{S}_{f_{k,1}}} Q_{f_{k,1}}^s(t)\mu_k^s(t)
\end{aligned} \tag{37}$$

Besides,

$$\begin{aligned}
&\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}_s\setminus\mathcal{F}_{in}} Q_f^s(t)\sum_{s'\in\mathcal{S}_{p(f)}} X_{p(f)}^{(s',s)}(t)B_{p(f)}^{s'}(t)\\
=&\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}_s\setminus\mathcal{F}_{in}}\sum_{s'\in\mathcal{S}_{p(f)}} Q_f^s(t)X_{p(f)}^{(s',s)}(t)B_{p(f)}^{s'}(t)\\
=&\sum_{(s',s)\in\mathcal{E}}\sum_{f\in\mathcal{F}_s\setminus\mathcal{F}_{in}} Q_f^s(t)\mathcal{I}_{p(f)\in\mathcal{F}_{s'}}X_{p(f)}^{(s',s)}(t)B_{p(f)}^{s'}(t)\\
=&\sum_{(s',s)\in\mathcal{E}}\sum_{f\in\mathcal{F}_s\setminus\mathcal{F}_{in}} Q_f^s(t)X_{p(f)}^{(s',s)}(t)B_{p(f)}^{s'}(t)\\
=&\sum_{(s',s)\in\mathcal{E}}\sum_{f\in\mathcal{F}_{nt}} Q_{n(f)}^s(t)X_f^{(s',s)}(t)B_f^{s'}(t)
\end{aligned} \tag{38}$$

where the indicator $\mathcal{I}_{p(f)\in\mathcal{F}_{s'}}=1$ if $p(f)\in\mathcal{F}_{s'}$ and 0 otherwise. Note that the last two equality holds because if $p(f)\notin\mathcal{F}_{s'}$, then $X_f^{(s',s)}(t)=0$ for all the time. Hence, we have

$$\begin{aligned}
&\Delta_V(\boldsymbol{Q}(t))\le B+C(\boldsymbol{Q}(t))\\
&+\mathbb{E}\left\{\sum_{k=1}^{K}\sum_{s\in\mathcal{S}_{f_{k,1}}}\left[-Q_k^p(t)+\alpha Q_{f_{k,1}}^s(t)\right]\mu_k^s(t)\middle|\boldsymbol{Q}(t)\right\}\\
&+\mathbb{E}\left\{\sum_{(s',s)\in\mathcal{E}}\sum_{f\in\mathcal{F}_{nt}}\Big[\alpha Q_{n(f)}^s(t)+Vw_{s',s}(t)\Big]\right.\\
&\left.X_f^{(s',s)}(t)B_f^{s'}(t)\middle|\boldsymbol{Q}(t)\right\}\\
&+\mathbb{E}\left\{\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}_s}\Big[V\gamma\boldsymbol{\lambda}_s^TY_f^s(t)-\alpha Q_f^s(t)\phi_f(Y_f^s(t))\Big]\middle|\boldsymbol{Q}(t)\right\}
\end{aligned} \tag{39}$$

By minimizing the upper bound of the drift-plus-penalty expression in (39), we can minimize the long-term time average of total system cost while stabilizing all processing queues. To transform the minimization of the above bound to maximization of the objective in (15), we have the following statement. We denote the objective function at time slot t by $J(t)$ and its respective optimal solutions by $\boldsymbol{\mu}^*$, $\boldsymbol{X}^*$, and $\boldsymbol{Y}^*$.

$$\begin{aligned}
J_t(\boldsymbol{\mu},\boldsymbol{X},\boldsymbol{Y})\triangleq&\sum_{k=1}^{K}\sum_{s\in\mathcal{S}_{f_{k,1}}}\left[-Q_k^p(t)+\alpha Q_{f_{k,1}}^s(t)\right]\mu_k^s\\
&+\sum_{(s',s)\in\mathcal{E}}\sum_{f\in\mathcal{F}_{nt}}\left[Vw_{s',s}(t)+\alpha Q_f^s(t)\right]B_{p(f)}^{s'}(t)X_f^{(s',s)}\\
&+\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}_s}\left[V\boldsymbol{\gamma}_s^TY_f^s-\alpha Q_{n(f)}^s(t)\,\phi_f\left(Y_f^s\right)\right]
\end{aligned} \tag{40}$$

and

$$\boldsymbol{\mu}^*,\boldsymbol{X}^*,\boldsymbol{Y}^*=\arg\min_{\boldsymbol{\mu},\boldsymbol{X},\boldsymbol{Y}} J_t(\boldsymbol{\mu},\boldsymbol{X},\boldsymbol{Y}) \tag{41}$$

Hence, for any other feasible scheduling decisions $\boldsymbol{X}$ and $\boldsymbol{Y}$ made during time slot t, we have

$$J_t(\boldsymbol{\mu},\boldsymbol{X},\boldsymbol{Y})\le J_t(\boldsymbol{X}^*,\boldsymbol{Y}^*) \tag{42}$$

By taking the conditional expectation on both sides conditional on $\boldsymbol{Q}(t)$, we have

$$\mathbb{E}\left\{J_t(\boldsymbol{\mu},\boldsymbol{X},\boldsymbol{Y})\middle|\boldsymbol{Q}(t)\right\}\le\mathbb{E}\left\{J_t(\boldsymbol{X}^*,\boldsymbol{Y}^*)\middle|\boldsymbol{Q}(t)\right\} \tag{43}$$

for any feasible $\boldsymbol{X}$ and $\boldsymbol{Y}$.

Inequality (43) reveals that $\boldsymbol{X}^*,\boldsymbol{Y}^*$ also maximize the conditional expectation of $J_t(\boldsymbol{X},\boldsymbol{Y})$, thusly minimizing the upper bound of drift-plus-penalty in (39). In such a way, instead of directly solving the long-term stochastic optimization problem (15), we can opportunistically choose a feasible association to solve the following problem during each time slot.

$$\begin{aligned}
\underset{\boldsymbol{\mu},\boldsymbol{X},\boldsymbol{Y}}{\text{Minimize}}\quad & J_t(\boldsymbol{\mu},\boldsymbol{X},\boldsymbol{Y})\\
\text{Subject to}\quad & \mu_k^s(t)\in\mathbb{Z}_+,\ \ \forall s\in\mathcal{S},k\in\{1,\dots,K\}\\
& X_f^s(t)\in\{0,1\},\ \ \forall s\in\mathcal{S},\ \ f\in\mathcal{F}_s\\
& Y_f^s(t)\in\mathbb{R}^n,\ \ \forall s\in\mathcal{S},\ \ f\in\mathcal{F}_s\\
& (2),(5),(6).
\end{aligned} \tag{44}$$

APPENDIX-B
PROOF

We assume there is an S-only algorithm achieves optimal time-average total cost (infimum) $m^*+\gamma g^*$ with action $\tilde{\boldsymbol{\mu}}(t)$, $\tilde{\boldsymbol{X}}(t)$, and $\tilde{\boldsymbol{Y}}(t)$, for $t=\{0,1,2,\cdots\}$.

From [21], we know that to ensure queue stability, the expectation of arrival rate must be no more than the expectation of service rate, *i.e.*, there exists some $\epsilon\ge0$ such that,

$$\begin{aligned}
&\mathbb{E}\left\{\sum_{k=1}^{K}A_k(t)\middle|\mathbf{Q}(t)\right\}+\epsilon\\
\le\ &\mathbb{E}\left\{\sum_{f\in\mathcal{F}}\sum_{s\in\mathcal{S}_f}\phi_f(\tilde{Y}_f^s(t))\middle|\mathbf{Q}(t)\right\}
\end{aligned} \tag{45}$$

Denote $\boldsymbol{\mu}'(t)$ $\boldsymbol{X}'(t)$, $\boldsymbol{Y}'(t)$ as the decisions over time, and $m'(t)$, $g'(t)$ as the corresponding costs given by P-OSCARS algorithm. Per (35), the one-slot drift-plus-penalty is

$$\begin{aligned}
\Delta_V(\mathbf{Q}(t))=&\,\mathbb{E}\left\{L(\mathbf{Q}(t+1))-L(\mathbf{Q}(t))\middle|\mathbf{Q}(t)\right\}\\
&+V\mathbb{E}\left\{m'(t)+\gamma g'(t)\middle|\mathbf{Q}(t)\right\}\\
\le&\,B+C(\mathbf{Q}(t))-\mathbb{E}\left\{\sum_{k=1}^{K}Q_k^p(t)\mu_k'(t)\middle|\boldsymbol{Q}(t)\right\}
\end{aligned}$$

$$+\alpha\mathbb{E}\left\{\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}_s\bigcap\mathcal{F}_{in}}Q_f^s(t)\left[{\mu'}_{k_f}^s(t)-\phi_f({Y'}_f^s(t))\right]\middle|\boldsymbol{Q}(t)\right\}$$
$$+\alpha\mathbb{E}\left\{\sum_{s\in\mathcal{S}}\sum_{f\in\mathcal{F}_s\setminus\mathcal{F}_{in}}Q_f^s(t)\left[\sum_{s'\in\mathcal{S}_{p(f)}}{X'}_f^{(s',s)}(t)B_{p(f)}^{s'}(t)-\right.\right.$$
$$\left.\left.\phi_f({Y'}_f^s(t))\right]\middle|\boldsymbol{Q}(t)\right\}+V\mathbb{E}\left\{m'(t)+\gamma g'(t)\middle|\boldsymbol{Q}(t)\right\}$$

Remind that m^* and g^* are the infimum of the achievable time-average total cost. Thereby substituting (45) into the above expression and adopting the definition (13), we obtain

$$\begin{aligned}&\mathbb{E}\left\{L(\mathbf{Q}(t+1))-L(\mathbf{Q}(t))\middle|\mathbf{Q}(t)\right\}+V\mathbb{E}\left\{m'(t)+\gamma g'(t)\middle|\mathbf{Q}(t)\right\}\\&\leq B+V\mathbb{E}\left\{m^*(t)+\gamma g^*(t)\middle|\mathbf{Q}(t)\right\}-\epsilon\mathbb{E}\left\{h(t)\middle|\mathbf{Q}(t)\right\}\end{aligned}\tag{46}$$

Taking expectation over both sides, we obtain

$$\begin{aligned}&\mathbb{E}\left\{L(\mathbf{Q}(t+1))\right\}-\mathbb{E}\left\{L(\mathbf{Q}(t))\right\}+V\mathbb{E}\left\{m'(t)+\gamma g'(t)\right\}\\&\leq B+V\mathbb{E}\left\{m^*(t)+\gamma g^*(t)\right\}-\epsilon\mathbb{E}\left\{h(t)\right\}\end{aligned}\tag{47}$$

Summing over $t=\{0,1,\cdots,T-1\}$, we have

$$\begin{aligned}&\mathbb{E}\left\{L(\mathbf{Q}(T-1))\right\}-\mathbb{E}\left\{L(\mathbf{Q}(0))\right\}+V\mathbb{E}\left\{\sum_{t=0}^{T-1}(m'(t)+\right.\\&\left.\gamma g'(t))\right\}\leq BT+V\mathbb{E}\left\{\sum_{t=0}^{T-1}(m^*(t)+\gamma g^*(t))\right\}-\epsilon\mathbb{E}\left\{h(t)\right\}\end{aligned}\tag{48}$$

By applying inequality (48), we show the $[O(V),O(1/V)]$ trade-off between the time-averages of total queue backlog size and total cost.

1) Dividing both sides of (48) by VT, rearranging items and neglecting non-positive quantities in right side, we obtain

$$\begin{aligned}&\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\{(m'(t)+\gamma g'(t))\right\}\\&\leq\frac{B}{V}+\frac{\mathbb{E}\{L(\mathbf{Q}(0))\}}{VT}+\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\{(m^*(t)+\gamma g^*(t))\right\}\end{aligned}\tag{49}$$

As $T\to\infty$, we have

$$\begin{aligned}&\limsup_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\{(m'(t)+\gamma g'(t))\right\}\\&\qquad\leq\limsup_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\{(m^*(t)+\gamma g^*(t))\right\}+\frac{B}{V}\end{aligned}\tag{50}$$

2) Similarly, by dividing both sides of (48) by ϵT, we get

$$\begin{aligned}&\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left\{h(t)\right\}\\&\leq\frac{B}{\epsilon}+\frac{\mathbb{E}\{L(\mathbf{Q}(0))\}}{\epsilon T}+\frac{V\sum_{t=0}^{T-1}\mathbb{E}\left\{(f^*(t)+\gamma g^*(t))\right\}}{\epsilon T}\end{aligned}\tag{51}$$

As $T\to\infty$, we obtain

$$\bar{h}\leq\frac{V}{\epsilon}\left(\bar{m^*}+\gamma\bar{g^*}\right)+\frac{B}{\epsilon}\tag{52}$$

■

REFERENCES

[1] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[2] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint vm placement and routing for data center traffic engineering," in *Proceedings of IEEE INFOCOM*, 2012.

[3] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of ACM SIGCOMM*, 2009.

[4] G. P. Katsikas, T. Barbette, D. Kostic, R. Steinert, and G. Q. Maguire Jr, "Metron: Nfv service chains at the true speed of the underlying hardware," in *Proceedings of USENIX NSDI*, 2018.

[5] J. Broughton, "Netflix adds download functionality," https://technology.ihs.com/586280/netflix-adds-download-support, 2016.

[6] 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.

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

[8] 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.

[9] A. Mohammadkhan, S. Ghapani, G. Liu, W. Zhang, K. Ramakrishnan, and T. Wood, "Virtual function placement and traffic steering in flexible and dynamic software defined networks," in *Proceedings of IEEE LANMAN*, 2015.

[10] K. Zhang, B. He, J. Hu, Z. Wang, B. Hua, J. Meng, and L. Yang, "G-net: Effective gpu sharing in nfv systems," in *Proceedings of USENIX NSDI*, 2018.

[11] A. Tootoonchian, A. Panda, C. Lan, M. Walls, K. Argyraki, S. Ratnasamy, and S. Shenker, "Resq: Enabling slos in network function virtualization," in *Proceedings of USENIX NSDI*, 2018.

[12] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[13] S. Woo, J. Sherry, S. Han, S. Moon, S. Ratnasamy, and S. Shenker, "Elastic scaling of stateful network functions," in *Proceedings of USENIX NSDI*, 2018.

[14] J. Zhang, W. Wu, and J. C. Lui, "On the theory of function placement and chaining for network function virtualization," in *Proceedings of ACM MobiHoc*, 2018.

[15] Q. Zhang, Y. Xiao, F. Liu, J. C. Lui, J. Guo, and T. Wang, "Joint optimization of chain placement and request scheduling for network function virtualization," in *Proceedings of IEEE ICDCS*, 2017.

[16] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Proceedings of IEEE INFOCOM*, 2015.

[17] Y. Sang, B. Ji, G. R. Gupta, X. Du, and L. Ye, "Provably efficient algorithms for joint placement and allocation of virtual network functions," in *Proceedings of IEEE INFOCOM*, 2017.

[18] A. Sheoran, P. Sharma, S. Fahmy, and V. Saxena, "Contain-ed: An nfv micro-service system for containing e2e latency," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 5, pp. 54–60, 2017.

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

[20] C.-L. Hsieh and N. Weng, "Nf-switch: Vnfs-enabled sdn switches for high performance service function chaining," in *Proceedings of IEEE ICNP*, 2017.

[21] 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.

[22] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, 2001.

[23] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: distributed, low latency scheduling," in *Proceedings of ACM SOSP*, 2013.

[24] L. Ying, R. Srikant, and X. Kang, "The power of slightly more than one sample in randomized load balancing," in *Proceedings of IEEE INFOCOM*, 2015.

[25] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of ACM SIGCOMM*, 2010.

[26] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers, randomly." in *Proceedings of USENIX NSDI*, 2012.
[27] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proceedings of ACM SIGCOMM*, 2008.
[28] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *Proceedings of IEEE CloudNet*, 2014.
[29] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *Proceedings of IEEE CloudNet*, 2015.
[30] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea, "Dynamic chaining of virtual network functions in cloud-based edge networks," in *Proceedings of IEEE NetSoft*, 2015.
[31] Y. Li and M. Chen, "Software-defined network function virtualization: A survey," *IEEE Access*, vol. 3, pp. 2542–2553, 2015.
[32] C. K. Chui, G. Chen *et al.*, *Kalman Filtering, 5th edition*. Springer, 2017.
[33] S. J. Taylor and B. Letham, "Forecasting at scale," *The American Statistician*, vol. 72, no. 1, pp. 37–45, 2018.
[34] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.
[35] L. Wang, Z. Lu, X. Wen, R. Knopp, and R. Gupta, "Joint optimization of service function chaining and resource allocation in network function virtualization," *IEEE Access*, vol. 4, pp. 8084–8094, 2016.
[36] J. D. Little, "A proof for the queuing formula: L= λ w," *Operations Research*, vol. 9, no. 3, pp. 383–387, 1961.
[37] M. S. Yoon and A. E. Kamal, "Nfv resource allocation using mixed queuing network model," in *Proceedings of IEEE GLOBECOM*, 2016.
[38] S. D'Oro, L. Galluccio, S. Palazzo, and G. Schembra, "Exploiting congestion games to achieve distributed service chaining in nfv networks," *IEEE JSAC*, vol. 35, no. 2, pp. 407–420, 2017.
[39] T. Lukovszki and S. Schmid, "Online admission control and embedding of service chains," in *Proceedings of SIROCCO*, 2015.
[40] X. Zhang, C. Wu, Z. Li, and F. C. Lau, "Proactive vnf provisioning with multi-timescale cloud resources: Fusing online learning and online optimization," in *Proceedings of IEEE INFOCOM*, 2017.
[41] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive vnf scaling and flow routing with proactive demand prediction," in *Proceedings of IEEE INFOCOM*, 2018.