Online VNF Chaining and Predictive Scheduling: Optimality and Trade-Offs

Xi Huang, *Member, IEEE*, Simeng Bian, *Student Member, IEEE*, Xin Gao, *Graduate Student Member, IEEE*, Weijie Wu, *Member, IEEE*, Ziyu Shao, *Senior Member, IEEE*, Yang Yang, *Fellow, IEEE*, and John C. S. Lui, *Fellow, IEEE, ACM*

***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 real-time decision making with limited information. Furthermore, the benefits of predictive scheduling to NFV systems still remain unexplored. In this article, we propose *POSCARS*, an efficient predictive and online service chaining and resource scheduling scheme that achieves tunable trade-offs among various system metrics with stability guarantee. Through a careful choice of granularity in system modeling, we acquire a better 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 to achieve the optimality with only limited information. By employing randomized load balancing techniques, we propose three variants of POSCARS to reduce the overheads of decision making. Theoretical analysis and simulations show that POSCARS and its variants require only mild-value of future information to achieve near-optimal system cost with an ultra-low request response time.**

Manuscript received August 19, 2019; revised February 26, 2020, April 17, 2020, October 19, 2020, and November 18, 2020; accepted April 6, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Mascolo. Date of publication April 22, 2021; date of current version August 18, 2021. This work was supported in part by the Nature Science Foundation of Shanghai under Grant 19ZR1433900, in part by the National Key Research and Development Program of China under Grant 2020YFB2104300, and in part by the National Development and Reform Commission of China (NDRC) under Grant "5G Network Enabled Intelligent Medicine and Emergency Rescue System for Giant Cities." The work of John C. S. Lui was supported in part by the GRF under Grant 14200420. The article was presented in part at the 2019 IEEE Global Communications Conference (GLOBECOM). *(Corresponding author: Ziyu Shao.)*

Xi Huang, Simeng Bian, Xin Gao, and Ziyu Shao are with the School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China (e-mail: huangxi@shanghaitech.edu.cn; biansm@shanghaitech.edu.cn; gaoxin@shanghaitech.edu.cn; shaozy@shanghaitech.edu.cn).

Weijie Wu is an independent researcher. He resides in San Mateo, CA 94403, USA (e-mail: wuwjpku@gmail.com).

Yang Yang is with the Shanghai Institute of Fog Computing Technology (SHIFT), ShanghaiTech University, Shanghai 201210, China, also with the Research Center for Network Communication, Peng Cheng Laboratory, Shenzhen 518000, China, and also with Shenzhen Smart City Technology Development Group Company Ltd., Shenzhen 518046, China (e-mail: yangyang@shanghaitech.edu.cn).

John C. S. Lui is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong (CUHK), Hong Kong (e-mail: cslui@cse.cuhk.edu.hk).

This article has supplementary downloadable material available at https://doi.org/10.1109/TNET.2021.3072423, provided by the authors.

Digital Object Identifier 10.1109/TNET.2021.3072423

***Index Terms*— NFV, service chaining, resource allocation, predictive scheduling.**

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 [35]. As a revolutionary technique, NFV paves the way for operators towards better manageability and quality-of-service of network services.

In NFV systems, each network service is implemented as 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. Considering the high cost of VNF migration and instantiation [24], VNF replacement can only be performed infrequently; hence, when it comes to flow-scale or request-scale operations, function placement can be viewed as a static operation. 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) [18], 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 be treated through, *a.k.a. service chaining*.

Given that VNF placement is considered static at the time scale of flow or request operations [27], 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 [17], but often times request traffic is highly fluctuating in both temporal and spatial dimensions [25]. In such cases, static schemes may lead to workload imbalance among instances, leaving some instances overloaded and others 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 [26]) have enabled real-time adjustment of resource allocation among various functions on the same server.

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

1) *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 which often times conflict with each other [46]. It is important to characterize their trade-offs to acquire a comprehensive understanding of our design space and tune the system towards the particularly desirable state.
2) *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 a distributed design is often preferred.
3) *Understanding Benefits of Predictive Scheduling:* A natural optimization of online decision making is about how to leverage recently developed machine learning techniques [5], [49] to predict future traffic information to reduce response time and improve quality-of-service. Particularly, in the past few years, predictive analysis and scheduling have been widely considered to promote the performance of NFV-based systems [28], [31], [43], [68]. Despite the rapid development of prediction-based approaches [14], [22], [23], [38], [65], it still remains open what are the fundamental benefits of predictive scheduling to NFV systems, even in the presence of prediction errors. Answers to the questions are the key to understanding whether the endeavor is worthy to put on predictive VNF scheduling, and whether one can tolerate the worst possible case that may occur.

Despite the recent headway on VNF scheduling [17], [18], 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 [37], 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 [65], then the decision making will inevitably incur a rather high computational overhead. Such issue not only prohibits a deep understanding on system dynamics, but also prevents us from obtaining efficient and accurate strategy design.

In this article, we overcome such difficulties by applying a number of novel techniques. Our contributions include:

Modeling and Formulation: We propose a novel model that separates the granularity of system state characterization and strategy making. In particular, we develop a queuing model at the request granularity to characterize system dynamics. Unlike flow-level abstraction, our model requires 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. As for strategy making, it is conducted at the granularity of request batch in a per-time-slot fashion to avoid the high overheads of per-request optimization. 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 our formulated long-term stochastic optimization problem into a series of sub-problems over time slots. By exploiting their unique structure, we propose *POSCARS*, a Predictive Online Service Chaining And Resource Scheduling scheme. Particularly, POSCARS includes two coupled parts. One is for the predictive scheduling of requests, while the other is for service chaining and resource allocation. The former part takes advantage of predicted information to effectively reduce request delays. The latter part can incur a near-optimal system cost while stabilizing all queues in the system. Furthermore, it can also achieve a tunable control between system cost optimization and queue stability.

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

Experiment Verification and Investigation: We conduct trace-driven simulations and our 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 an ultra-low request response time.

The rest of this article is organized as follows. In Section II, we show a motivating example of predictive scheduling in NFV systems. Section III presents our model and formulation, followed by the design and performance analysis of POSCARS and its variants in Section IV. We show simulation results and analysis in Section V, then review related work in Section VI. Finally, Section VII concludes this article.

II. Motivating Example

In this section, we first show a motivating example that exhibits the potential trade-off in the multi-objective optimization for different system metrics, including reduction in energy costs and communication costs, as well as shortening response times 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, pre-generated, and pre-served by the

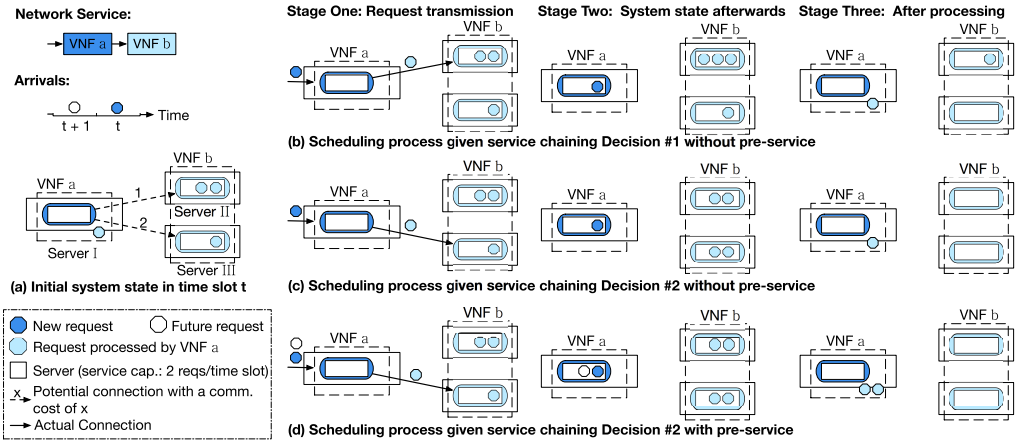


Fig. 1. The system evolution in time slot t under different service chaining decisions, with or without pre-service. *Basic settings:* There is one network service with two VNFs, *i.e.*, VNF a and VNF b. VNF a has one instance, while VNF b has two instances. Every instance maintains one queue to buffer 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.

system.[1] Figure 1(a) shows the basic settings and initial system state in time slot t. All VNF instances are readily deployed on servers with a fixed placement. Each instance maintains a queue to buffer any untreated request. 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 are not counted in the queues, but readily to be sent to VNF b's instances in the next time slot; 2) requests that have been 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, the total communication cost, and the total residual backlog size at the end of time t. [2] Figures 1(b) - 1(d) compare the scheduling processes under different service chaining decisions.

In Figure 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 loads among VNF b's instances. Note 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 loads among 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. Decision #2 incurs *a higher total cost of* 7 *on energy and communication, but with no residual backlogs*.

Insight 1: Figures 1(b) and 1(c) show that we cannot 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 length.

Additionally, we find that server I is under-utilized in both Figures 1(b) and 1(c), because VNF a's instance only receives and handles the new request at time t. In fact, Figure 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. In other words, 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 cost 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

[1] As a leading streaming service provider, Netflix has been actively migrating its streaming services onto NFV-based platforms (e.g., by Red Hat Inc.) to deliver high-quality streaming services. In such a scenario, the packets (deemed as requests for network services) are sent through particular chains of VNFs such as load balancing, encoding/decoding, encryption/decryption, etc., before their deliveries to end devices. By employing machine learning techniques to predict user preferences and network state variations, Netflix predicts demanded packets of different streaming subscriptions, then proactively pre-delivers them onto end devices [6].

[2] By applying *Little's law* [32], a short queue length implies a short queueing delay or a short response time.

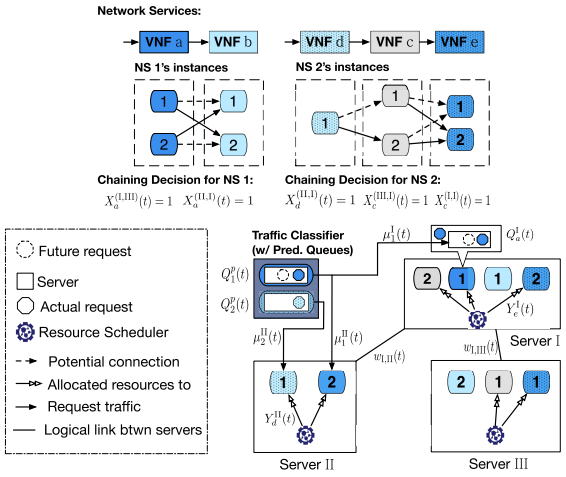


Fig. 2. An instance of our system model. There are two network services (NS 1 and NS 2) with their VNF instances deployed on servers I, II, and III. At the beginning of time slot t, the traffic classifier admits and pushes requests to the queues $Q_1^p(t)$ and $Q_2^p(t)$ with respect to their requested services. For each instance, based on its server's resource scheduling, it serves requests from its processing queue and forwards requests to its next VNF instances, *e.g.*, instance of VNF a on server I to instance of VNF b on server III.

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; *i.e.*, each request is assumed to have equal size and require the same amounts of computation to be processed. We show an instance of our system model in Figure 2 and summarize main notations in Table I. More details of service chaining can be found in IETF RFC-7665 [15].

A. Substrate Network Model

We consider the substrate network with a set $\mathcal{S}$ of heterogeneous servers. On each server s, we consider R types of resources, *e.g.*, GPU [62], CPU cores [26], and cache [50]. 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 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 cost $[w_{s',s}(t)]_{s',s}$ 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 [16]. 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.

TABLE I
MAIN NOTATIONS

Substrate Network Model	
$\mathcal{S}$	The set of servers that host VNFs
$w_{s',s}(t)$	Communication cost of sending one request from server s' to server s
$c_{s,i}$	The capacity of type-i resource type on server s
$\lambda_{s,i}$	The unit cost of type-i resource on server s
Network Service Model	
K	Number of network services
$\mathcal{L}_k$	Chain length of network service k
$\mathcal{F}$	The set of virtual network functions (VNFs)
$\mathcal{F}_{in}$	The set of all ingress VNFs
$\mathcal{F}_{nt}$	The set of all non-terminal VNFs
$f_{k,j}$	The j-th VNF in network service chain k
k_f	The network service that contains VNF f
$p(f)$	Previous VNF of f in service chain k_f
$n(f)$	Next VNF of f in service chain k_f
$\theta_{f,i}$	Number of requests processed by unit of type-i resource
Deployment Model	
$\mathcal{S}_f$	The set of all servers that host f's instances
$\mathcal{F}_s$	The set of VNFs with instances residing on server s
System Dynamics	
$A_k(t)$	Number of new requests for network service k arriving in time t
$Q_k^{(d)}(t)$	Number of untreated requests for network service k in the next d-th slot from time t
$Q_k^p(t)$	The prediction queue length for network service k in time t
$Q_f^s(t)$	Queue length of VNF f's instance on server s in time t
$B_f^s(t)$	Number of processed requests by VNF f's instance on server s in time $(t-1)$, and to be sent to the next VNF in time t
$\delta_k(t)$	Total number of admitted requests for network service k
$\delta_k^{(d)}(t)$	Number of admitted requests from $Q_k^{(d)}(t)$
Scheduling Decisions	
$\mu_k^s(t)$	Number of admitted requests for service k onto server s
$X_f^{(s',s)}(t)$	$X_f^{(s',s)}(t) = 1$ if the instance of $n(f)$ on server s is selected to receive the processed requests from f's instance on server s' and zero otherwise
$Y_f^s(t)$	the vector of allocated resources on server s VNF f's instance
System Objectives	
$m(t)$	Total communication cost in time t
$g(t)$	Total computation cost in time t
$h(t)$	Weighted total queue length in time t

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 each 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 is 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 [54]. We assume that each VNF has at most one instance on each server but it can have multiple instances on different servers. The placement of VNF instances is assumed to be pre-determined by adopting VNF placement schemes similar to existing ones [9], [44], [61], [64]. Depending on the placement, the instances required by each service are not necessarily readily available on the same server. Note that our model can be further extended to cases with each VNF having multiple instances on the same server.

For VNF $f \in \mathcal{F}$, we use $\mathcal{S}_f$ to denote the set of servers that host f's instances. Correspondingly, each server s hosts a subset $\mathcal{F}_s \subseteq \mathcal{F}$ of VNFs. Every instance maintains one queue to buffer its relevant requests. For example, if VNF f has one instance on server s, then the instance has a queue 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 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 costs due to additional rescheduling. The choice depends on the trade-off made by system designers. Here we adopt the queueing 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. For such cases, our model captures the system dynamics at a finer granularity than the flow-level abstraction, and can be further extended to handle correlations between requests.

Next, considering the ever-increasing interests of applying predictive scheduling to promote performances of NFV systems in the recent years [28], [31], [43], [68],[3] we focus on the case where a system can predict and pre-serve future request arrivals for network services in a finite number of time slots ahead. Though the techniques and analysis of prediction are still under active development [38], [41], [65], we do not assume any particular prediction technique in this article.[4] 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. 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 that buffers untreated future requests for network service k. In practice, the prediction queues can be hosted on servers or storage systems in proximity to the request traffic classifier [47]. To simplify notations, we use $\boldsymbol{Q}(t)$ to denote the vector of all queues' length $\{Q_k^p(t)\}_{k=1}^K$ and $\{Q_f^s(t)\}_{s\in\mathcal{S}, f\in\mathcal{F}_s}$.

E. System Workflow and Scheduling Decisions

System Workflow: At the beginning of each 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 [26]. The instances then process the requests from their respective queues. At the end of time slot, the prediction window moves one slot ahead.

In the above process, we need to consider three kinds of scheduling decisions.

i) Admission Decision: For every network service, the traffic classifier decides the number of untreated newly arriving and future requests, to be admitted 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

[3]To accommodate the requirements of ultra-high processing speed and ultra-low latency for NFV, existing works mainly resort to the analysis of system logs and effective use of sampled queueing dynamics and structural information of VNF service chains to reduce prediction overheads without compromising prediction accuracies. Meanwhile, in the industry, as far as we are concerned, more and more efforts have been devoted by multiple parties (*e.g.*, telecom operators) in the recent years to the development of predictive scheduling in NFV-based systems, so as to promote QoE and reduce costs.

[4]Although only few concrete solutions have been disclosed about prediction in real-world NFV systems, considering the rapid advancement of 5G techniques, machine learning, and dedicated chip design, the development of such predictive techniques for NFV systems with stringent requirements (*e.g.*, even higher data rates) is just a matter of time. Moreover, we believe that in the near future, more and more companies such as Netflix and Facebook will migrate their services onto NFV-based systems and enjoy the benefit from predictive scheduling in terms of QoE improvement and cost reduction.

$Q_k^p(t)$, *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 [20]. 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,

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

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 $\boldsymbol{\mu}(t)$.

ii) 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 zero 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 $X_f^{(s,s')}(t) = 0$ in each time slot t. Note that dynamic request steering can be implemented by adopting VNFs-enabled SDN switches [19]. We denote all chaining decisions by $\boldsymbol{X}(t)$.

iii) Resource Scheduling Decision: 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 amount of allocated resources should not exceed server s's resource capacity, *i.e.*,

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

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 cannot process too many requests and 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 the 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, 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. 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. Accordingly, given $\mu_k(t)$, prediction queue $Q_k^p(t)$ is updated as follows

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

With the above workflow, we have the following 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 $s \in \mathcal{S}_f$. Accordingly, the update function for queue length $Q_f^s(t)$ 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. Then the queueing update function is given by

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

where $B_{p(f)}^{s'}(t) \triangleq \phi_{p(f)}\big(Y_{p(f)}^{s'}(t-1)\big)$, *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 example of our queue model, in which there are two network services that require six types of VNF whose instances are hosted on three servers. Each of the network services has a prediction window of size two. In Figure 3,

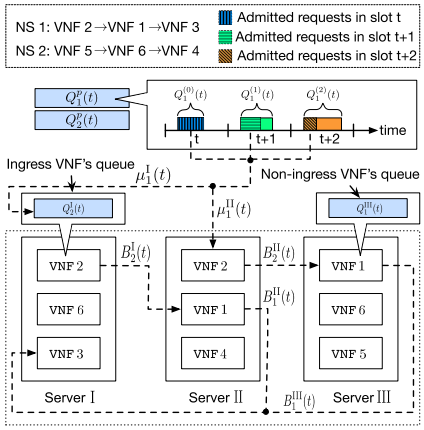


Fig. 3. An instance of queueing model with a lookahead window size of two.

we show 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 $X_2^{(\mathrm{I,II})} = X_2^{(\mathrm{II,III})} = X_1^{(\mathrm{II,I})} = X_1^{(\mathrm{III,I})} = 1$.

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 cost 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

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

where $w_{s',s}(t)$ denotes the communication cost of transferring a request between servers s' and s in time t. Then the total communication cost in time t is given by

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

Energy Cost: Efficient resource utilization for servers is another important objective to achieve in NFV systems [56]. Given the resource allocation $Y_f^s(t)$, we define the corresponding energy 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

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

Queue Stability: Considering the responsiveness of requests and scarcity of computational resources such as memory and cache, it is also imperative to ensure that no queues would be overloaded. We denote the weighted total queue length 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 measures the importance of stabilizing instances queues compared to prediction queues. Accordingly, we define the queue stability [39] 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 costs and energy costs while ensuring queue stability. With such formulation, we seek to achieve the potential trade-off among different system metrics.

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

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

IV. Algorithm Design and Performance Analysis

In this section, 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 [39] to transform the long-term stochastic optimization problem into a series of sub-problems over time slots, as specified by the following lemma.

*Lemma 1: By applying Lyapunov optimization techniques and the concept of opportunistically minimizing an expectation, problem **P1** can be transformed to the following optimization problem to be solved in each time slot t:*

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

$$\begin{aligned} & \text{Subject to } (2), (5), (6) \text{ and } \mu_k^s \in \mathbb{Z}_+, \ \forall k, s \in \mathcal{S}_{f_{k,1}} \\ & \qquad X_f^{(s',s)} \in \{0,1\}, \ \forall s', s \in \mathcal{S}, \ f \in \mathcal{F}_s \\ & \qquad Y_f^s \in \mathcal{O}_f \ \forall s \in \mathcal{S}, f \in \mathcal{F}_s. \end{aligned} \tag{17}$$

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

$$l_f^{(s',s)}(t) \triangleq \left[V w_{s',s}(t) + \alpha Q_{n(f)}^s(t)\right] B_f^{s'}(t), \tag{18}$$

such that V is a positive parameter that measures the importance of minimizing system cost compared to stabilizing system queues, and $r_f^s(t, Y)$ is defined as

$$r_f^s(t, Y) \triangleq V \gamma \boldsymbol{\lambda}_s^T Y - \alpha Q_f^s(t) \phi_f(Y). \tag{19}$$

The detailed proof of Lemma 1 is relegated to Appendix-A of the supplementary materials. Here we provide a sketch of how the problem transformation is carried out. Note that the key technique we adopt is the *drift-plus-penalty* method [39], which generally aims to stabilize a queueing network while also optimizing the time-average of some objective (*e.g.*, the total cost of energy consumption and communication in **P1**). To this end, a quadratic function (*a.k.a.* Lyapunov function) is first introduced to characterize the stability of all queues in each time slot. Then the key idea of the method is to introduce a drift-plus-penalty term to characterize the joint change in the queue stability and the objective value across time slots. In particular, the drift-plus-penalty term is defined as the weighted sum of two parts. One is defined as the difference (*a.k.a.* drift) between the Lyapunov functions of two consecutive time slots, which measures the short-term change in queue stability. The other part is defined as the instant objective value in a time slot. Then the stability of the queueing network and the optimization of the time-average of

the objective are jointly achieved by deriving an online control policy that greedily minimizes the upper bound of the drift-plus-penalty term during each time slot. In this way, it can be proven that it is equivalent to solving problem **P1** by resolving a series of subproblems (**P2**) over time slots.

Note that by solving problem **P2** over time slots, problem **P1** can be solved asymptotically optimally as the total number of time slots T and the value of parameter V both approach infinity, as shown by Theorem 1 in Sec. IV-B. Furthermore, problem **P2** can be decomposed into three sub-problems for request admission, service chaining, and resource allocation, with their decisions in each time slot denoted by $\boldsymbol{\mu}$, $\boldsymbol{X}$, and $\boldsymbol{Y}$, respectively. Specifically, based on Lemma 1, we propose POSCARS, a predictive online service chaining and predictive resource scheduling scheme, and show its pseudocode in Algorithm 1.

Remark 1: Regarding request admission, when all instances are severely loaded compared to the prediction queue. In order not to overload any instances, POSCARS admits only untreated requests at the current time slot and spreads them evenly onto the least loaded instances. Otherwise, i.e., when instances all have shorter queue lengths than the prediction queue, POSCARS admits all future requests and assigns them to the least loaded instances.

Remark 2: POSCARS decides the service chaining by jointly considering instances' queue length and the communication cost. Recall by the definition in (18), the weighted summation $\alpha Q^s_{n(f)}(t) + V w_{s',s}(t)$ 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. Besides, a large communication cost $w_{s',s}(t)$ also makes it less willing to choose the target instance.

Remark 3: On server s, the resource allocation is decided by jointly considering the resource cost and the queue length of its resident instances. Particularly, we regard the term $V\gamma\boldsymbol{\lambda}_s - \alpha Q^s_f(t)\boldsymbol{\phi}_f$ 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.*, $V\gamma\lambda_{s,i} - \alpha Q^s_f(t)\phi_f$, it is a weighted difference between the unit cost $\lambda_{s,i}$ of type-i resource and the queue length $Q^s_f(t)$ of the instance. A high unit resource cost will result in a prudent allocation. On the other hand, a sufficiently long queue length 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.

Remark 4: In practice, there is often a tradeoff between the accuracy of predictive scheduling and the incurred overhead such as the control traffic overhead among VNF instances and the overhead for acquiring real-time information for decision-making. The finer the granularity of scheduling, the more accurate the request processing will be.

B. Performance Analysis

We analyze the computational complexity of POSCARS in each time slot as follows. For each network service, it takes $O(|\mathcal{S}|)$ time to make request admission decisions (lines 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 (lines 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. In practice, POSCARS can be run in a distributed manner. Particularly, the request admission sub-routine can be implemented on each traffic classifier with a computational complexity of $O(K \times |\mathcal{S}|)$; meanwhile, the service chaining and resource scheduling sub-routines can be deployed on the hypervisor of each server, with computational complexities of $O(|\mathcal{S}|)$ for each instance and $O(\Omega_{max} \times |\mathcal{F}|)$ for each server, respectively. Note that Ω_{max} is the maximum number of applicable resource allocations.

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 cost $\{\boldsymbol{\lambda}_s\}$, and communication cost $\boldsymbol{w}(t)$. 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**

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 length and total cost via the tunable parameter V. In particular, given the value of γ, let $(m^* + \gamma g^*)$ denote the optimal value of problem **P1**; then we have the following theorem.

Theorem 1: Suppose that $h(0) < \infty$ and, given the system resource capacities on each server and VNF placement, there exists an online scheme which ensures that, for each VNF instance, the mean arrival rate is smaller than its mean service rate. Under POSCARS without prediction, there exist constants

$B > 0$ and $\epsilon > 0$ such that

$$\limsup_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left\{m(t)+\gamma g(t)\right\} \leq \frac{B}{V} + m^* + \gamma g^*,$$

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

The proof is relegated to Appendix-B of the supplementary materials. Theorem 1 demonstrates an $[O(V), O(1/V)]$ trade-off between system cost optimization and queue stability. Particularly, without prediction, POSCARS can achieve a near-optimal cost within an $O(1/V)$ optimality gap but at the cost of an $O(V)$ increase in the time-averaged total queue length. Intuitively, with a large value for V, VNF instances are more willing to steer requests to their successive instances in nearby servers, while server would allocate resources to instance with less energy cost. As a result, the total cost can be effectively reduced; however, some servers may become hot spots and the total queue length will increase. In contrast, a smaller value of V conduces to more balanced queue loads among servers and more energy cost consumed to serve requests, leading to an increasing total cost. Moreover, given predicted information about future requests, POSCARS can achieve a better trade-off with a notable delay reduction by pre-serving requests with surplus system resources. We verify such advantages by our simulation results in Section V.

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 each 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** [36], **Batch-Sampling** [40], and **Batch-Filling** [57].

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 of 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 remains. To mitigate such issues, we adopt the idea of *batch-sampling* [40] and *batch-filling* [57] and propose another two variants of POSCARS, namely *POSCARS with Batch-Sampling* (P-BS) and *POSCARS with Batch-Filling* (P-BF), respectively. Basically, these two variants split the processed requests on each instance into batches, each batch with a size of b, then carry out scheduling upon such request batches. 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 elaborate on the design of P-BS and P-BF 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 [4], 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 Poisson request arrivals at 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 [48] and Fat-Tree [3]. Both topologies have a comparable scale to clusters in data center networks, each equipped with 720 switches, 24 servers with deployed VNFs, and the remaining 3456 servers as hosts that generate service requests. Particularly, in Fat-Tree, there are 24 pods, each pod containing to 144 servers. Among such servers in each pod, 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 a 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 [1], [7], [34]. 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 loads 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 ranging 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 remaining VNFs of each service are chosen uniformly from other 30 commonly-used VNFs [30] at random without replacement. For each VNF, the total number of instances ranges from 12 to 18.

Prediction Settings: Network services' traffics often have different predictabilities. 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 *false-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) [8]; 2) distribution estimator (Distr), which generates the next estimate by independent sampling from the distribution of arrivals learned from historical data; 3) Prophet (FB) [49], Facebook's time-series forecasting procedure; 4) moving average (MA) and 5) exponentially weighted moving average (EWMA) [5].

Baseline Schemes: We compare POSCARS with three baseline schemes, including *Random*, *JSQ* (Join-the-Shortest-Queue), and state-of-the-art *OneHop-SCH* (OneHop scheduling) [52]. These schemes differ in the service chaining strategy from POSCARS. In Random scheme, each instance uniformly randomly sends requests to one of its successors. In JSQ scheme, each instance sends requests to its least-loaded successor. In OneHop-SCH, each instances sends requests to its successor with the least communication cost and idle capacity.

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 as it incurs 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. If a request is pre-served before it arrives, then the system is assumed to respond to the request instantly upon its arrival, and the request will experience a zero response time.

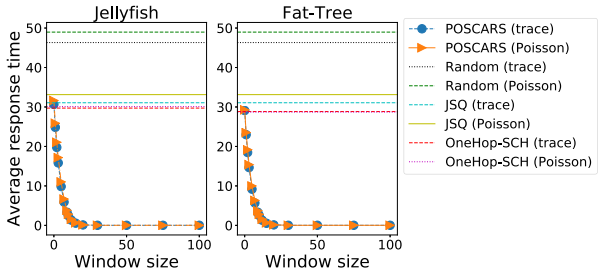
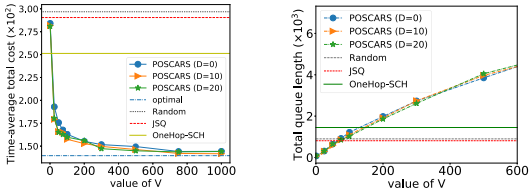


Fig. 4. Average response time ($m.s$) with various window sizes given trace and Poisson arrival process, under different topologies.



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

Fig. 5. Total queue length under different window sizes.

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. Therefore, the essential benefits of predictive scheduling come from the load balancing in the temporal dimension. To verify such an intuition, we first consider the case with perfectly predicted request 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 by the baseline schemes remain constant since they do not involve predictive scheduling. Random incurs the highest response time ($\sim$ 47ms), as it disregards information about workloads or communication costs when scheduling 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 a comparable performance with OneHop-SCH; but as the value of 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 the value of 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 (a large value of 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 only present results under Fat-Tree and trace-driven request loads.

Backlog-Cost Trade-Off With Parameter V: Recall from Theorem 1 that the value of parameter V controls the trade-off between the cost optimality gap and the total queue backlog size. Figures 5(a) and 5(b) verify such a trade-off. Figure 5(a) compares the time-average communication cost of POSCARS with $D = 0$, 10, 20, against baselines. Both Random and JSQ incur a high total cost since their decision making disregards the resultant communication cost and the heterogeneity of servers in terms of energy cost. OneHop-SCH achieves a lower total cost by about 13.6%, by taking its advantages of jointly optimizing cost and shortening queue lengths based on flow-level statistics. Given different values of D, POSCARS achieves a close-to-optimal time-average total cost as the value of V rises up to 10^3. Notably, POSCARS excels OneHop-SCH whenever $V > 10$.

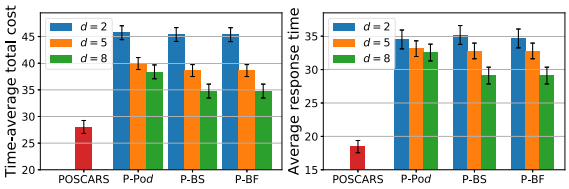
However, recall that parameter V weighs the importance of minimizing system cost compared to maintaining queue stability. Hence, to reduce system cost, large values of V also lead to increased queue lengths. By *Little's theorem* [32], this would increase the response time as well. In Figure 5(b), we see that the total queue length 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 cost and shorter queue lengths. In practice, such an interval may vary from system to system but it is usually proportional to the ratio of magnitudes of the total queue length to the total system cost.

POSCARS and Its Variants: When forwarding requests, POSCARS requires each instance to collect statistics from 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 [57] from $O(n)$ to $O(1)$, where n denotes the total number of candidate instances. Figure 6 evaluates the total cost 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 cost and response time than P-Pod, because aggregated sampling is often more conducive to lowering the cost [40].

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. 6. Comparison among POSCARS and its variants.

C. Performance Evaluation Under Imperfect Prediction

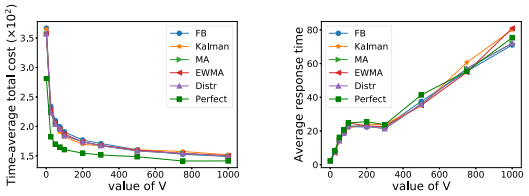
In practice, prediction errors are inevitable due to dataset bias and noise. To explore the fundamental limits of predictive scheduling, we evaluate the impact of imperfect prediction on the system performance.

Total Cost and Response Time vs. V: Figure 7 compares the time-average total cost and average response time incurred 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 false-negative and false-positive predictions. Particularly, the system pre-allocates extra resources to pre-serve false-positive requests, resulting in higher total cost. Figure 7(b), shows the overall ascending trend proportional to the value of V. This is due to that larger values of V lead to a greater total queue length, and by *Little's theorem* [32], a greater queue length implies longer response time. However, we also see that, even under imperfect prediction, predictive scheduling does not necessarily lead to a longer response time than that under perfect prediction.

To figure out the reason, we consider two extreme cases. One is all-*false-negative*, *i.e.*, during each time slot, all future request arrivals in the lookahead window are false-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 some extra requests are wrongly predicted to arrive.

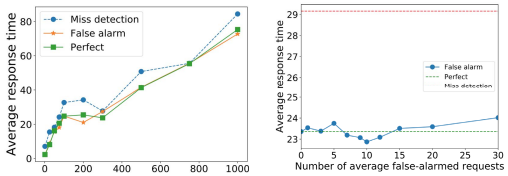
Perfect Prediction vs. Two Extremes: Figure 8(a) compares average response times under perfect prediction and the two extremes, with D=5, α=10, and 5 false-positive requests on average. Overall, the average response time is proportional to the value of V. Miss detection incurs a higher response time 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 each 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 the request queues' length, thus leading to longer response times. The *second* is that, according to lines 5 - 9 in Algorithm 1, false-positive requests result in a greater prediction queue length. 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 $D = 5$,



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

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



(a) Different values of V (b) Different false-alarms

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

$\alpha = 10$, 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 admissions, 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 extends the total queue length, offsetting and eventually outweighing the effect of load balancing.

Insight: Imperfect prediction does not necessarily degrade system performance in terms of 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

In this section, we first summarize existing works that study the optimization of NFV from different aspects. Then we narrow down our focus onto those that are most relevant to this article and compare their proposed approaches with ours.

A. Optimizing NFV/VNF From Different Aspects

A wide range of recent works have studied NFV systems from various aspects. Below we take a brief overview and discuss how they are related to our work.

- ◇ *VNF placement*: In NFV, the placement of VNF instances often has a significant impact on system performances [27] and thus deserves an elaborate design. A number of existing works have been conducted to this end (*e.g.*, [2], [10], [12], [42], [63]). In practice, such approaches can serve to decide the VNF placement, upon which our schemes can carry out their scheduling procedures accordingly.
- ◇ *VNF Resource Allocation*: Another series of works (*e.g.*, [26], [29], [67]) focused on the optimization of resource allocation for VNF/NFV, with the aim to minimize VNF execution overheads and accelerate the processing speed of VNF instances. They concentrated on achieving such improvements with particular hardware designs. Different from such works, we mainly focus on exploiting predicted information to perform effective scheduling on existing NFV systems. Nonetheless, our schemes can be applied to systems built with their solutions.
- ◇ *Load Balancing*: Existing works (*e.g.*, [51], [53], [59]) also developed various schemes to balance the workloads among chained VNF instances to improve resource utilization and fault tolerance while shortening delays in NFV systems. In practice, existing solutions can serve as reference points for system designers to tune the proper value of parameter V for desired performance metrics.
- ◇ *Performance Characterization:* Another line of works have devoted their efforts to characterizing various dynamics of NFV systems such as performance interferences among VNF instances [45], [60]. Insights from such works can be combined with our schemes to achieve even better performance.

B. Chaining and Resource Scheduling of VNFs in NFV

Regarding the optimization of VNF service chaining and resource scheduling in NFV, existing works 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.* [61] considered the joint optimization for VNF placement and service chaining. They formulated the problem as an ILP problem and developed an efficient rounding-based approximation algorithm with performance guarantee. Yoon *et al.* [58] adopted the BCMP queueing model for VNF service chains and proposed heuristics to approximately minimize the expected waiting time of service chains. Wang *et al.* [52] considered the joint optimization of service chaining and resource allocation and devised a greedy scheme that aims to place instances and schedule traffic with minimum link cost, CAPEX, and OPEX. Later, D'Oro *et al.* [11] studied service chaining problem from the perspective of congestion games. By formulating the problem as an atomic weighted congestion game, they proposed a distributed algorithm that provably converges to the Nash equilibrium. On the other hand, Zhang *et al.* [64] formulated a request-level optimization problem based on steady-state metrics and proposed 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 assume 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 and decentralized manner.

Of the second category are the online schemes that process requests upon their arrivals. Under this setting, Mohammadkhan *et al.* [37] formulated the VNF placement for service chaining as an MILP problem based on flow abstraction and designed a heuristic to solve the problem incrementally. Lukovszki *et al.* [33] proposed an online algorithm that performs request admission and service chaining with a logarithmic competitive ratio. Zhang *et al.* in [66] devised a novel VNF brokerage service model and online algorithms

to predict traffic demands, purchase VMs and deploy VNFs. Further, Fei *et al.* [13] presented 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. Later, Xiao *et al.* [55] designed an adaptive service chaining deployment scheme based on deep reinforcement learning techniques, which conducts service chaining to serve incoming requests in an online fashion. Such 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. In comparison, our model considers such trade-offs and separates the granularity of system state and decision making. Besides, we also explore the fundamental benefits and limits of predictive scheduling, which remains open in NFV systems.

VII. Conclusion

In this article, 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 a near-optimal system cost while effectively shortening average response time. Our results also show that prediction with mild false-positive conduces to shorter response times. In addition, note that the fair sharing of resources and performance isolation among VNF instances is the key to maintaining high quality of service. Therefore, it is an interesting direction for future work to establish a more effective joint service chaining and scheduling scheme with multi-resource fairness consideration among VNF instances. Moreover, it would also be intriguing to explore the interplay between resource fairness and other performance metrics.

References

[1] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *Proc. IEEE 4th Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2015, pp. 171–177.

[2] S. Agarwal, F. Malandrino, C.-F. Chiasserini, and S. De, "Joint VNF placement and CPU allocation in 5G," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Apr. 2018, pp. 1943–1951.

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

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

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

[6] Netflix Inc. *Learning How to Entertain the World*. Accessed: Mar. 2019. [Online]. Available: https://research.netflix.com/research-area/machine-learning

[7] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea, "Dynamic chaining of virtual network functions in cloud-based edge networks," in *Proc. 1st IEEE Conf. Netw. Softwarization (NetSoft)*, Apr. 2015, pp. 1–5.

[8] C. K. Chui and G. Chen, *Kalman Filtering*. Berlin, Germany: Springer, 2017.

[9] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 1346–1354.

[10] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic, latency-optimal vNF placement at the network edge," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Apr. 2018, pp. 693–701.

[11] S. D'Oro, L. Galluccio, S. Palazzo, and G. Schembra, "Exploiting congestion games to achieve distributed service chaining in NFV networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 2, pp. 407–420, Feb. 2017.

[12] X. Fei, F. Liu, H. Xu, and H. Jin, "Towards load-balanced VNF assignment in geo-distributed NFV infrastructure," in *Proc. IEEE/ACM 25th Int. Symp. Qual. Service (IWQoS)*, Jun. 2017, pp. 1–10.

[13] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive VNF scaling and flow routing with proactive demand prediction," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Apr. 2018, pp. 486–494.

[14] X. Gao, X. Huang, S. Bian, Z. Shao, and Y. Yang, "PORA: Predictive offloading and resource allocation in dynamic fog computing systems," *IEEE Internet Things J.*, vol. 7, no. 1, pp. 72–87, Jan. 2020.

[15] J. Halpern *et al.*, *Service Function Chaining (SFC) Architecture*, document IETF RFC-7665, 2015.

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

[17] H. Hantouti, N. Benamar, T. Taleb, and A. Laghrissi, "Traffic steering for service function chaining," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 487–507, 1st Quart., 2019.

[18] J. Gil Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 3, pp. 518–532, Sep. 2016.

[19] C.-L. Hsieh and N. Weng, "NF-switch: VNFs-enabled SDN switches for high performance service function chaining," in *Proc. IEEE 25th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2017, pp. 1–6.

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

[21] X. Huang, S. Bian, X. Gao, W. Wu, Z. Shao, and Y. Yang, "Online VNF chaining and scheduling with prediction: Optimality and trade-offs," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2019, pp. 1–6.

[22] X. Huang, S. Bian, Z. Shao, and H. Xu, "Predictive switch-controller association and control devolution for SDN systems," *IEEE/ACM Trans. Netw.*, vol. 28, no. 6, pp. 2783–2796, Dec. 2020.

[23] X. Huang, Z. Shao, and Y. Yang, "POTUS: Predictive online tuple scheduling for data stream processing systems," *IEEE Trans. Cloud Comput.*, early access, Oct. 20, 2020, doi: 10.1109/TCC.2020.3032577.

[24] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM placement and routing for data center traffic engineering," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 2876–2880.

[25] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: Measurements & analysis," in *Proc. 9th ACM SIGCOMM Conf. Internet Meas. Conf. (IMC)*, 2009, pp. 202–208.

[26] 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 *Proc. USENIX NSDI*, 2018, pp. 171–186.

[27] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1409–1434, 2nd Quart., 2019.

[28] S. Lange, H.-G. Kim, S.-Y. Jeong, H. Choi, J.-H. Yoo, and J. W.-K. Hong, "Machine learning-based prediction of VNF deployment decisions in dynamic networks," in *Proc. 20th Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Sep. 2019, pp. 1–6.

[29] X. Li, X. Wang, F. Liu, and H. Xu, "DHL: Enabling flexible software network functions with FPGA acceleration," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2018, pp. 1–11.

[30] Y. Li and M. Chen, "Software-defined network function virtualization: A survey," *IEEE Access*, vol. 3, pp. 2542–2553, 2015.

[31] Z. Li *et al.*, "Predictive analysis in network function virtualization," in *Proc. IMC*, Oct. 2018, pp. 161–167.

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

[33] T. Lukovszki and S. Schmid, "Online admission control and embedding of service chains," in *Proc. SIROCCO*, 2015, pp. 104–118.

[34] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *Proc. IEEE 3rd Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2014, pp. 7–13.

[35] 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 Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 1st Quart., 2016.

[36] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 10, pp. 1094–1104, Oct. 2001.

[37] A. Mohammadkhan, S. Ghapani, G. Liu, W. Zhang, K. K. Ramakrishnan, and T. Wood, "Virtual function placement and traffic steering in flexible and dynamic software defined networks," in *Proc. 21st IEEE Int. Workshop Local Metrop. Area Netw.*, Apr. 2015, pp. 1–6.
[38] S. Nanda, F. Zafari, C. DeCusatis, E. Wedaa, and B. Yang, "Predicting network attack patterns in SDN using machine learning approach," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2016, pp. 167–172.
[39] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synth. Lectures Commun. Netw.*, vol. 3, no. 1, pp. 1–211, Jan. 2010.
[40] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in *Proc. ACM SOSP*, Nov. 2013, pp. 69–84.
[41] I. S. Petrov, "Mathematical model for predicting forwarding rule counter values in SDN," in *Proc. IEEE Conf. Russian Young Researchers Electr. Electron. Eng. (EIConRus)*, Jan. 2018, pp. 1313–1317.
[42] C. Pham, N. H. Tran, S. Ren, W. Saad, and C. S. Hong, "Traffic-aware and energy-efficient vNF placement for service chaining: Joint sampling and matching approach," *IEEE Trans. Services Comput.*, vol. 13, no. 1, pp. 172–185, Jan. 2020.
[43] F. Rath *et al.*, "SymPerf: Predicting network function performance," in *Proc. SIGCOMM Posters Demos*, Aug. 2017, pp. 34–36.
[44] 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 *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, May 2017, pp. 1–9.
[45] M. Savi, M. Tornatore, and G. Verticale, "Impact of processing-resource sharing on the placement of chained virtual network functions," *IEEE Trans. Cloud Comput.*, early access, May 2, 2019, doi: 10.1109/TCC.2019.2914387.
[46] S. Schneider, S. Draxler, and H. Karl, "Trade-offs in dynamic resource allocation in network function virtualization," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2018, pp. 1–3.
[47] A. Sheoran, P. Sharma, S. Fahmy, and V. Saxena, "Contain-ed: An NFV micro-service system for containing e2e latency," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 47, no. 5, pp. 54–60, 2017.
[48] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers, randomly," in *Proc. USENIX NSDI*, 2012, pp. 225–238.
[49] S. J. Taylor and B. Letham, "Forecasting at scale," *Amer. Statistician*, vol. 72, no. 1, pp. 37–45, 2018.
[50] A. Tootoonchian *et al.*, "ResQ: Enabling SLOs in network function virtualization," in *Proc. USENIX NSDI*, 2018, pp. 283–297.
[51] H. Wang and J. Schmitt, "Load balancing–towards balanced delay guarantees in NFV/SDN," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2016, pp. 240–245.
[52] 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.
[53] T. Wang, H. Xu, and F. Liu, "Multi-resource load balancing for virtual network functions," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 1322–1332.
[54] S. Woo, J. Sherry, S. Han, S. Moon, S. Ratnasamy, and S. Shenker, "Elastic scaling of stateful network functions," in *Proc. USENIX NSDI*, 2018, pp. 299–312.
[55] Y. Xiao *et al.*, "NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning," in *Proc. IEEE/ACM IWQoS*, Jun. 2019, pp. 1–10.
[56] Z. Xu, F. Liu, T. Wang, and H. Xu, "Demystifying the energy efficiency of network function virtualization," in *Proc. IEEE/ACM IWQoS*, Jun. 2016, pp. 1–10.
[57] L. Ying, R. Srikant, and X. Kang, "The power of slightly more than one sample in randomized load balancing," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 692–722.
[58] M. S. Yoon and A. E. Kamal, "NFV resource allocation using mixed queuing network model," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–6.
[59] C. You and L. M. Li, "Efficient load balancing for the VNF deployment with placement constraints," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–6.
[60] C. Zeng, F. Liu, S. Chen, W. Jiang, and M. Li, "Demystifying the performance interference of co-located virtual network functions," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Apr. 2018, pp. 765–773.
[61] J. Zhang, W. Wu, and J. C. S. Lui, "On the theory of function placement and chaining for network function virtualization," in *Proc. 18th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, Jun. 2018, pp. 91–100.
[62] K. Zhang *et al.*, "G-net: Effective GPU sharing in NFV systems," in *Proc. USENIX NSDI*, 2018, pp. 187–200.
[63] Q. Zhang, F. Liu, and C. Zeng, "Adaptive interference-aware VNF placement for service-customized 5G network slices," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Apr. 2019, pp. 2449–2457.
[64] Q. Zhang, Y. Xiao, F. Liu, J. C. S. Lui, J. Guo, and T. Wang, "Joint optimization of chain placement and request scheduling for network function virtualization," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 731–741.
[65] S. Zhang, L. Huang, M. Chen, and X. Liu, "Proactive serving decreases user delay exponentially," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 43, no. 2, pp. 39–41, Sep. 2015.
[66] X. Zhang, C. Wu, Z. Li, and F. C. M. Lau, "Proactive VNF provisioning with multi-timescale cloud resources: Fusing online learning and online optimization," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, May 2017, pp. 1–9.
[67] Y. Zhang, Z.-L. Zhang, and B. Han, "HybridSFC: Accelerating service function chains with parallelism," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2019, pp. 1–7.
[68] L. M. M. Zorello, M. G. T. Vieira, R. A. G. Tejos, M. A. T. Rojas, C. Meirosu, and T. C. M. de Brito Carvalho, "Improving energy efficiency in NFV clouds with machine learning," in *Proc. IEEE 11th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2018, pp. 710–717.