# Adaptive VNF Scaling and Flow Routing with Proactive Demand Prediction

Xincai Fei[1] Fangming Liu*[1] Hong Xu[2] Hai Jin[1]
[1]Key Laboratory of Services Computing Technology and System, Ministry of Education,
School of Computer Science and Technology, Huazhong University of Science and Technology
[2]NetX Lab @ City University of Hong Kong

***Abstract*—With the evolution of Network Function Virtualizaiton (NFV), enterprises are increasingly outsourcing their network functions to the cloud. However, using virtualized network functions (VNFs) to provide flexible services in today's cloud is challenging due to the inherent difficulty in intelligently scaling VNFs to cope with traffic fluctuations. To best utilize cloud resources, NFV providers need to dynamically scale the VNF deployments and reroute traffic demands for their customers. Since most existing work is reactive in nature, we seek a proactive approach to provision new instances for overloaded VNFs ahead of time based on the estimated flow rates. We formulate the VNF provisioning problem in order that the cost incurred by inaccurate prediction and VNF deployment is minimized. In the proposed online algorithm, we first employ an efficient online learning method which aims at minimizing the error in predicting the service chain demands. We then derive the requested instances with adaptive processing capacities and call two other algorithms for new instance assignment and service chain rerouting, respectively, while achieving good competitive ratios. The joint online algorithm is proven to provide good performance guarantees by both theoretical analysis and trace-driven simulation.**

## I. Introduction

Compared to traditional hardware-based middleboxes, the recent trend of *network function virtualization* (NFV) promises to implement network functions in virtual machines (VMs) running on commodity servers [1]. In NFV, a *service chain* is an abstracted view of a service that specifies the set of required *virtualized network functions (VNFs)*, as well as the order in which they must execute [2]. NFV can significantly reduce captial expenses and increase the flexibility through automated management.

Encouraged by the benefits, enterprises start to use NFV to transform their hardware middlebox infrastructure. A recent initiative proposes that packet processing can benefit from outsourcing to the cloud [3]. Along this line of thinking, we envision that third-party companies or cloud providers themselves may act as NFV service providers, to create and deploy VNF instances, and set service chain routing paths for their customers (*e.g.*, enterprises, home and mobile users) on demand. Recently, NEC/Netcracker has unveiled the NFV cloud marketplace by introducing a Network-as-a-Service solution [4], which enables the rapid commercialization of network services.

Outsourcing packet processing to the cloud is increasingly becoming viable, as customers may only experience ∼1 ms latency increase on average and a bandwidth inflation of 3.8%, according to the results in [3]. Besides, after outsourcing reliant network functions to the cloud, customers can then center on their core business and leave the complex scheduling and provisioning tasks to the NFV providers.

In this paper, we focus on the NFV marketplace where an NFV provider rents or owns cloud resources to provide software packet processing services for customers, predicts upcoming traffic demands, reprovisions the VNF instances and aggregates service chains to better utilize the limited and valuable resources over time. With prediction, VNF instances can be dynamically scaled up or down. There are several challenges to enable this vision. (i) Effective demand prediction approach needs to be developed since either under- or over-provisioning has negative effects on the NFV provider's profit. (ii) Which types of VNF instance in the chain should be scaled with adaptive processing capacity due to varying processing times of different network functions [5]. (iii) Where to assign the newly launched VNF instances to better serve the traffic. (iv) Which links should the flows be routed to those new VNF instances, with the link bandwidth constraints.

However, existing scheme [6], [7] with dynamic resource provisioning largely ignores the above assignment and routing problems. Once instances are launched, it has almost been taken for granted that traffic flows traversing VNFs in a sequence can obtain enough bandwidth. In [7], a graph neural network based approach is exploited to predict future resource requirements for VNFs, which might not work well since VNFs of different types usually show variance in resource requirements [8] even they are in the same service chain. Another scheme [9] scales VNF resources vertically on-the-fly, which is not encouraged in most cases as it requires rebooting the system [5]. Further, existing literature about VNF scaling [10], [5] is reactive in nature, and may degrade the service quality since spinning up new VNF instances (*e.g.*, copying VM images) incurs delay and packet loss.

We propose a proactive online algorithm called **V**NF **P**rovisioning for **C**ost **M**inimization ($VPCM$) for the NFV

*This work was supported in part by the National Key Research & Development (R&D) Plan under grant 2017YFB1001703, in part by NSFC under Grant 61761136014 (NSFC-DFG) and 61722206 and 61520106005, in part by the National 973 Basic Research Program under Grant 2014CB347800, and in part by the Fundamental Research Funds for the Central Universities under Grant 2014YQ001, 2014TS006 and 2017KFKJXX009. (Corresponding author: Fangming Liu)

provider, to tackle the above challenges. The cost here is incurred by inaccurate prediction and VNF deployment. The former consists of the cost due to the traffic loss or waiting to be served in case of under-provisioning, or the cost caused by resource wastage in case of over-provisioning. The latter is incurred mainly from copying VNF images to a VM when launching a new VNF instance. We predict the flows rates of service chain demands rather than resource requirements of VNFs, as supported by a recent measurement [11]. With prediction results, we derive the processing capacity requested by VNFs. Then, we simultaneously introduce vertical scaling by allocating adaptive processing capacities to the new instances when performing horizontal scaling (installation of VNF instances) ahead of time.

To minimize the former cost, we minimize the prediction errors. In particular, we employ an efficient online learning method called follow-the-regularized-leader (FTRL) to predict the upcoming flows. Since it is expensive to directly solve the problem with the original loss function, we carefully seek a surrogate loss function instead. An upper bound of the regret of FTRL is then proven, when compared with the best static predictor. When launching new instances, we allocate each new instance either with a best-fit capacity or with the maximum capacity [12] (adaptive scaling), based on the derived processing capacity requested for each type of VNFs. This not only saves cloud resources, but also decreases the number of VNF instances to be launched.

Next, we need to assign these new instances on the network servers with enough space capacity and route flows along service chains to these new instances on appropriate network links. The strategies for new instance assignment and service chain routing should be adjusted accordingly, as the residual capacity of each server and link vary over time. While cross-chain instance sharing and flows splitting are permitted [13], we design two online algorithms that are called by *VPCM* in sequence, to solve the above two problems, respectively. We regard the assignment problem as a variable-sized bin packing problem, and propose an efficient heuristic that never exceeds $\frac{3}{2}$ the optimal. For service chain routing, the proposed primal dual algorithm has objective function value within $(1+\epsilon)$-ratio of the optimal. An overall performance ratio is then derived, which jointly considers the regret in predicting flow rates and the competitiveness of new instance deployment with the guidance of the best static demand predictor.

To summarize, our main contributions are as follows:

- Towards an NFV marketplace, we present an optimization formulation of the VNF provisioning problem that minimizes the cost incurred by inaccurate prediction and VNF deployment, in the offline setting.
- We propose an online algorithm to proactively provision VNF instances, including both new instance assignment and flow routing. Instead of predicting resource requirements that might not be valid [7], we directly predict the flow rates of service chains, by embracing the online learning algorithms. Meanwhile, we adopt an adaptive scaling strategy by introducing vertical scaling when launching new instances ahead of time.
- We provide theoretical analysis for prediction regret and the performance of the proposed algorithms. Besides, by using trace-driven simulation, we confirm that the proposed *VPCM* algorithm achieves better prediction outcomes and less overall cost, compared with other comparison proposals.

## II. Related Work

With the rapid development of NFV, a large number of recent works studied VNF dynamic scaling, optimal placement, and service chain routing. Addis *et al.* [14] present a VNF chaining and placement model and devise a mixed integer linear programming formulation. Zhang *et al.* [15] propose a solution for the joint VNF chain placement and request scheduling problem, which maximizes the average resource utilization and minimizes the average response latency of each request. Fei *et al.* [16] present a framework of VNF assignment across geo-distributed NFV infrastructure for load balancing, in both central office and server level. Kuo *et al.* [17] design a service chain deployment algorithm, by jointly considering the relation between the link and server usage. Eramo *et al.* [9] investigate the resource consolidation problem when placing VNF instances, taking into account the reconfiguration costs due to the QoS degradation. They adopt vertical scaling techniques but cannot avoid the service outages.

The above literature focuses on offline deployment of NFV and ignores the traffic fluctuations. There are also a few studies dealing with dynamic resource provisioning for VNFs. Wang *et al.* [10] design online algorithms for VNF provisioning cost minimization, in case of both one service chain and multiple service chains. Ghaznavi *et al.* [5] optimize the placement of VNF instances in response to on-demand workload. These studies are reactive in nature. The closest approaches to our work are [6] and [7]. Mijumbi *et al.* [7] propose a graph neural network based algorithm by exploiting topology information of service chains, to predict future resource requirements for each VNF component. However, its predictive accuracy cannot be guaranteed since VNFs might have totally different resource requirements even though they are in a same topology. The authors in [6] fuse online learning and online optimization, and design a proactive online algorithm to purchase short- and long-term VMs for running VNF instances, while the service chain routing problem is ignored.

The online learning method FTRL [18] has excellent performance in solving convex optimization problems [19] with good sparsity. It cannot be directly applied to our problem since it is difficult to solve the linear loss function. Instead, we seek a surrogate loss function while maintaining sufficient accuracy. Variable-sized bin packing [20] and primal-dual algorithm [21] have been extensively researched. In our case, we carefully use these techniques to solve the VNF assignment and flow routing problems for newly launched instances in each time slot.

## III. Problem Model and Formulation

### *A. Cloud Network, VNFs and Service Chain Demands*

We model a cloud network as a directed graph $G = (V, E)$. Each node $v \in V$ may attach one or more NFV servers, and its space capacity is denoted as $C_v$, *i.e.*, the maximum capacity can support for VNF instances. A link $e \in E$ has a bandwidth capacity $B_e$, *i.e.*, the amount of available bandwidth.

The NFV service provider deploys and operates $N$ types of VNFs for customers in a number of $T$ time slots (*i.e.*, scaling intervals), each on the order of seconds or minutes. A total number of $I$ service chain demands arrive during $\mathcal{T} = \{1, 2, \cdots, T\}$. A demand $i \in \mathcal{I} = \{1, 2, \cdots, I\}$ arriving in $t_i$ is a 3-tuple $\langle o_i, s_i, \alpha_i(t)\rangle$ and lasts $\Delta t_i$ time slots for using service chain $i$, where $o_i \in V$ is the source node, $s_i \in V$ is the destination node and $\alpha_i(t)$ is the traffic rate of demand $i$ in $t$ for $t_i \leq t \leq t_i + \Delta t_i$, *i.e.*, $\alpha_i(t) = 0$ for $t \in [0, t_i) \cup (t_i + \Delta t_i]$. The traffic rate of a demand varies over time. As VNFs are a given sequence in the chain, let $\theta_n^i$ indicate whether type $n \in \mathcal{N} = \{1, 2, \cdots, N\}$ VNF is used ($\theta_n^i = 1$) in service chain $i$ or not ($\theta_n^i = 0$). The packet processing time of VNF $n$ is denoted with $\tau_n$, which embodies the VNF diversity in processing traffic.

When new type-$n$ VNF instances are launched, let $\phi_n^v$ denote whether these instances are deployed on server node $v$ ($\phi_n^v \geq 1$, as one or more type-$n$ VNF instances can be deployed on a same node) or not ($\phi_n^v = 0$). Meanwhile, the residual space capacity of node $v$ is changed over time and denoted by $C_v(t)$ in $t$. The residual capacity of a node can be computed according to the creation or release of VNF instances on that node. For example, the space capacity of node $v$ is $C_v(1) = C_v - \sum_{n\in\mathcal{N}} \phi_n^v c_n(1)$, where $c_n(1)$ is the capacity of type-$n$ VNF instance in $t = 1$ and the sum term means the total occupied capacity on node $v$ in $t = 1$. Similarly, the residual bandwidth capacity of link $e \in E$ is denoted by $B_e(t)$ in $t$.

### *B. VNF Provisioning and Flow Routing*

We consider that in initial state of $t = 0$, the VNFs requested by the previous demands have been placed/mapped in the server nodes with requested processing capacity $C_n(0)$ for each $n \in \mathcal{N}$. The bandwidth requirements for these demands can also be satisfied. However, the processing capacity of a VNF, as well as the bandwidth requirements, may not support a demand in successive time slots, due to the ebb and flow of the traffic rate and arrival of new demands.

Changing compute or memory resources on-the-fly is not supported once a VNF instance is created [5]. To ensure more efficient resource utilization, we adopt an adaptive scaling strategy for VNF instances renewal. That means, unlike most of horizontal scaling approaches with a same scaling capacity, the newly created VNF instances can be adjusted to a best-fit processing capacity, to cover the demands. This scaling is performed ahead of time rather than on-the-fly, rebooting system causing service outages can thus be avoided.

In time slot $t$, if the flow rate $\alpha_i(t)$ along each existing service chain $i$ is known, the processing capacity (number of packets processed per unit time) requested by VNF $n$ can be computed as follows [9]:

$$C_n(t) = \sum\nolimits_{i\in\mathcal{I}:t\in[t_i,t_i+\Delta t_i]} \theta_n^i \alpha_i(t)\tau_n/L_p, \tag{1}$$

where $L_p$ is the packet length in the flow. Let $c_n^{max}$ denote the maximum processing capacity of a type-$n$ VNF instance, which is defined as the maximum amount of traffic flows that a VNF instance can process [12] and can be easily converted to be the same unit as $C_n(t)$. Then, the number of newly launched instances of VNF $n$ in $t \in \mathcal{T}$ is

$$x_n^{new}(t) = \begin{cases} 0 & \text{if } C_n(t) \leq C_n(t-1), \\ \lceil \frac{C_n(t)-C_n(t-1)}{c_n^{max}} \rceil & \text{otherwise,} \end{cases} \tag{2}$$

where $C_n(t-1)$ is the total processing capacities of all type-$n$ VNF instances in $t-1$. That is, if $C_n(t) \leq C_n(t-1)$, no new type-$n$ instances need to be launched. If $0 < C_n(t) - C_n(t-1) \leq c_n^{max}$, one instance of VNF $n$ with capacity $C_n(t) - C_n(t-1)$ will be launched, denoted by $c_n^l(t)$; else $x_n^{new}(t) - 1$ instances with capacity $c_n^{max}$ and one instance with $C_n(t) - C_n(t-1) - (x_n^{new} - 1)c_n^{max}$ then will be launched, in $t$, if it exists. The latter capacity is denoted by $c_n^g(t)$.

For flow routing along a service chain, after a new VNF instance is created, flows of the demand can be split among multiple instances of this VNF, and aggregate the flow at the next hop (*i.e.*, successor VNF) [13]. Variable $y_i(e)$ defines the fraction of flow demand $i$ along link $e$, $\forall e \in E$. Meanwhile, the total bandwidth carried by the links along the path should be lower than their residual capacity. Flow routing algorithm for a service chain demand see Sec. V-B.

### *C. Online Decisions and Cost Minimization Problem*

For profit maximization, the NFV provider deploys VNFs in a proactive fashion. In time slot $t$, the NFV provider predicts the upcoming flow rate $\alpha_i(t+1)$ along each existing service chain $i$. Based on prediction outcomes and new arrival demands in $t+1$, the NFV provider examines whether existing VNFs have sufficient capacity to support the fluctuating and new demands. If not, the total processing capacity requested by a certain VNF and the number of newly launched instances of that VNF can thus be calculated according to (1) and (2), respectively. The provider then deploys $x_n^{new}(t+1)$ new instances with adaptive capacities for each VNF $n$ on available nodes along a path, which prepares for serving flows in $t+1$.

Due to inaccurate prediction, the pre-deployed VNF instances may still have not enough capacity to serve the flows due to under-provisioning. As a result, the provider will suffer revenue loss in that some flows have to wait to be served or are just directly dropped. For over-provisioning, the provider has to spend more self-cost on cloud resources due to wasting of extra resources. Both cases have negative effects on the provider's profit. Besides, when launching a new VNF instance, a deployment cost is always incurred for copying VM image that contains a certain VNF.

The provider aims at minimizing the overall loss incurred by two cases above and the deployment cost of VNFs over $T$

time slots. Let $\alpha_i^*(t)$ be the actual flow rate of demand $i$ in $t$, based on the actual arrival of this demand. $C_n^*(t)$ denotes the actual requested processing capacity for VNF $n$ computed by (1) using $a_i^*(t)$ . In case of under-provisioning ($\alpha_i(t) < \alpha_i^*(t)$), let $s(t)$ be the revenue loss of per one Gbit of traffic loss in $t$, then the total cost of revenue loss is $\sum_{t\in\mathcal{T}}\sum_{i\in\mathcal{I}} s(t)(\alpha_i^*(t) - \alpha_i(t))$, where $\alpha_i(t)$ is the flow rate predicted in $t-1$. In case of over-provisioning ($\alpha_i(t) > \alpha_i^*(t)$), let $u(t)$ denote the cost of per unit processing capacity in $t$, then the total cost caused by resource wastage is $\sum_{t\in\mathcal{T}}\sum_{n\in\mathcal{N}} u(t)(C_n(t) - C_n^*(t))$, where $C_n(t)$ is the requested processing capacity of VNF $n$ according to (1) using $\alpha_i(t)$. For deployment cost of VNFs, we first need to compute the number of instances for each VNF type running at $t$. Since we consider that VNFs have been placed in the server nodes in initial state of $t = 0$, the number of type-$n$ instances $x_n(0)$ is a known value. So the total number of type-$n$ instances in $t$ can be computed as follows:

$$x_n(t) = \begin{cases} x_n(0) & t = 0, \\ x_n(t-1) + x_n^{new}(t) & 1 \le t \le T. \end{cases} \tag{3}$$

If the deployment cost for type-$n$ instance in $t$ is $d_n(t)$, the total cost of deploying VNF instances can be denoted by $\sum_{t\in\mathcal{T}}\sum_{n\in\mathcal{N}} d_n(t)x_n(t)$.

Therefore, the overall cost minimization problem is:

$$\begin{aligned} \text{minimize} \quad & \sum_{t\in\mathcal{T}}\sum_{i\in\mathcal{I}} s(t)[\alpha_i^*(t) - \alpha_i(t)]^+ \\ & + \sum_{t\in\mathcal{T}}\sum_{n\in\mathcal{N}} \big\{u(t)[C_n(t) - C_n^*(t)]^+ + d_n(t)x_n(t)\big\} \end{aligned} \tag{4}$$

subject to:

$$(1), (2), (3),$$

$$\sum_{n\in\mathcal{N}} \{c_n^l(t), c_n^{max} + c_n^g(t)\}\phi_n^v \le C_v(t), \quad \forall v \in V, t \tag{5}$$

$$\sum_{i\in\mathcal{I}: t\in[t_i, t_i+\Delta t_i]} \alpha_i(t)y_i(e) \le B_e(t), \quad \forall e \in E, t \tag{6}$$

$$\sum_{v\in V} \phi_n^v = x_n(t), \quad \forall n \in \mathcal{N}, t \tag{7}$$

$$\begin{aligned} & \phi_n^v \in \{0,1\}, y_i(e), \alpha_i(t) \ge 0, x_n(t) \in \mathbb{Z}_+, \\ & \qquad \forall v \in V, e \in E, i \in \mathcal{I}, \quad n \in \mathcal{N}, t \end{aligned} \tag{8}$$

The notation $[\cdot]^+$ in objective function (4) is defined as $[a]^+ = \max\{a, 0\}$. Constraint (5) ensures that the total processing capacity of VNF instances assigned to node $v \in V$ cannot exceed the residual space capacity of that node in $t$. Here if $x_n^{new}(t) = 1$, $\{c_n^l(t), c_n^{max} + c_n^g(t)\} = c_n^l(t)$, since there just exists one instance with capacity $c_n^l(t)$; else, $\{c_n^l(t), c_n^{max} + c_n^g(t)\} = c_n^{max} + c_n^g(t)$, since there only exist $c_n^{max}$ and $c_n^g(t)$ instances, and the number of $c_n^{max}$ instances assigned to $v$ is determined by $\phi_n^v$. Constraint (6) guarantees that the total bandwidth carried by network link $e \in E$ is lower than its residual capacity $B_e(t)$ in $t$. Constraint (7) establishes the equality relation between type-$n$ instances deployed on all nodes and the number of type-$n$ instances running at $t$.

Since traffic changing effects and dependency relations of VNFs have been investigated in [22], we assume that each type of VNF has a traffic change ratio of 1, and our formulation can also be easily extended to enable different change ratios.

## IV. Proactive Online Algorithm with Adaptive VNF Scaling

In this section, we propose a proactive online algorithm, called **V**NF **P**rovisioning for **C**ost **M**inimization (*VPCM*), from which the NFV provider can learn to scale VNF instances with adaptive capacities to handle workload fluctuations in each $t \in \mathcal{T}$, with the help of effective demand prediction.

### A. Preliminaries

Since inaccurate prediction is inevitable, we first explain how to deal with the following two cases.

In case of over-provisioning, we don't directly shut down an extra VNF instance to avoid frequent creation or release of VNF instances under traffic fluctuations. Instead, we maintain a logical buffer queue to temporarily tag unused VNF instances for each type of VNF. The logical queue is not a real queue that occupies any specific server node. Specifically, a VNF instance will stay put once it is created in a scaling interval $t$, then the algorithm will tag it with $t$ and enqueue it to the tail of the queue. In under-provisioning case, buffered instances from the tail of the respective queue will be popped out and serve for those requested service chains, which further decreases the deployment cost. Unused VNF instances in the queue are removed after $\kappa$ time slots, which can be tuned by the NFV provider. The type-$n$ VNF instances tagged with the same scaling interval $t$ will be removed simultaneously.

### B. Demand Prediction Using FTRL

To provide effective prediction on traffic demand, we employ an online learning method called FTRL [18]. It performs well either on large or real-world dataset. To characterize traffic rate fluctuation, we assume $\alpha_i^{max} = \max_{t\in[t_i, t_i+\Delta t_i]} \alpha_i(t)$ for demand $i$, which is provided by customers based on prior experience. All predicted rates $\alpha_i(t)$ of demand $i$ are in $[0, \alpha_i^{max}]$. Based on this, we construct a loss function on variable $\alpha_i(t) \in [0, \alpha_i^{max}]$:

$$f_{it}(\alpha_i(t)) = |\alpha_i(t) - \alpha_i^*(t)|, \forall i \in \mathcal{I}, t \in \mathcal{T}, \tag{9}$$

to represent the prediction error in $t$. Note that $f_{it}(\cdot)$ is a convex function on variable $\alpha_i(t)$. A cost loss minimization problem over all time slots then can be expressed as follows, $\forall i \in \mathcal{I}$:

$$\min_{\alpha_i(t)\in[0,\alpha_i^{max}], \forall t\in\mathcal{T}} \sum_{t\in\mathcal{T}} f_{it}(\alpha_i(t)). \tag{10}$$

FTRL chooses the predicted rate value $\alpha_i(t+1) \in [0, \alpha_i^{max}]$ for demand $i$ that minimizes the cumulative loss so far on the previous $t$ time slots, and plus the rate value that minimizes the regularizer, *i.e.*,

$$\alpha_i(t+1) = \arg\min_{z_i\in[0,\alpha_i^{max}]} \sum_{s=1}^{t} f_{is}(z_i) + R(z_i), \tag{11}$$

where $z_i$ is a variable that refers to possible predicted rate for demand $i$ in $t+1$ and $R(z_i)$ is a convex regularization function (*i.e.*, the regularizer) with respect to variable $z_i$.

Solving this optimization problem at each time slot $t$ is expensive in general, which will destroy the spirit of online learning. Therefore, we seek a surrogate loss function $h_{it}(z_i)$ that satisfies: (i) the surrogate loss function is easy to be solved and has analytical solution; (ii) the gap between the solutions of the surrogate and original loss function cannot be too large. Then we covert (11) into the following problem:

$$\alpha_i(t+1) = \arg\min\nolimits_{z_i \in [0, \alpha_i^{max}]} h_{it}(z_i), \ \forall i \in \mathcal{I}. \tag{12}$$

However, how do we choose the the surrogate loss function that satisfies the two conditions above? Based on a unified analysis that matches or improves the best-ever known bounds in [18], given that $f_{it}(\alpha_i(t))$ is a convex function, we adopt the following surrogate loss function:

$$h_{it}(z_i) = \sum_{s=1}^{t} g_{is} \cdot z_i + r_{it}(z_i), \ \forall i \in \mathcal{I}, t \in \mathcal{T}, \tag{13}$$

where $g_{is}$ is the gradient of $f_{is}(\alpha_i(s))$, *i.e.*, $g_{is} = \nabla f_{is}(\alpha_i(s))$. According to (9), we have $\nabla f_{is}(\alpha_i(s))^2 = 1$. The second term in $h_{it}(z_i)$ is the regularization function corresponding to $R(z_i)$. $\forall z_i \in [0, \alpha_i^{max}]$, we have,

$$r_{it}(z_i) = \frac{1}{2} \sum_{s=1}^{t} \left( \frac{1}{\eta_s} - \frac{1}{\eta_{s-1}} \right) (z_i - \alpha_i(s))^2, \tag{14}$$

where $\eta_s = \Theta(\frac{1}{\sqrt{s}})$ is the learning rate in time slot $s$, which is a non-increasing sequence changing over time. Specifically, the learning rate $\eta_t = \frac{\beta}{\sqrt{\sum_{s=1}^{t} g_{is}^2}} = \frac{\beta}{\sqrt{t}}$ in $t$, where $\beta$ is a parameter depending on the features and dataset [23], and $\beta = \alpha_i^{max}$ is a good choice is our case as it characterizes the variation range of the traffic rate.

After the transformation of loss function, problem (12) is a smooth minimization problem with no constraints. Since $z_i \geq 0$, $h_{it}(z_i)$ is derivable on $z_i$. We just take the derivative of $h_{it}(z_i)$ about $z_i$ and let it to be equal to 0, *i.e.*, $\partial h_{it}(z_i) = 0$. The equation is easy to be solved and the solution $z_i$ is exactly the optimal predicted rate of $\alpha_i(t+1)$ [23].

### *C. Regret Analysis for Demand Prediction with FTRL*

Now we analyse the performance of FTRL in predicting traffic rates $\alpha_i(t+1)$, by computing the regret bound after the conversion to the surrogate loss function. Compared with the best static predictor $z_i^* \in [0, \alpha_i^{max}]$ obtained by the strategy in [24] with full knowledge of traffic rates, $\forall i \in \mathcal{I}$, the regret is computed as follows:

$$\text{Regret}_i^T(z_i^*) = \sum_{t=1}^{T} h_{it}(z_i) - \sum_{t=1}^{T} h_{it}(z_i^*). \tag{15}$$

**Lemma 1.** *The conversion from (11) to (12) is to apply FTRL on a linear function in which $f_{it}(z_i) = g_{it} \cdot z_i$ with a regularization function $r_{it}(z_i)$, where an online mirror descent (OMD) algorithm is derived. OMD achieves the same regret bound as FTRL compared with the best static predictor $z_i^*$, $\forall i \in \mathcal{I}$ and satisfies,*

$$\text{Regret}_i^T(z_i^*) \leq r_{iT}(z_i^*) + \frac{1}{2} \sum_{t=1}^{T} g_{it}^2 \eta_t. \tag{16}$$

The proof can be derived from [19] along almost the same line in analysing the regret bound of FTRL and OMD, which is omitted here due to space limitation.

**Theorem 1.** *The regret of $VPCM$ in predicting service chain demands, with respect to the best static prediction strategy that use $z_i^*$ for all $t \in \mathcal{T}$, is as follows:*

$$\text{Regret}^T(z_i^*) = \sum_{i \in \mathcal{I}} \text{Regret}_i^T(z_i^*) \leq \frac{3\sqrt{T} - 1}{2} \sum_{i \in \mathcal{I}} \alpha_i^{max}. \tag{17}$$

Detailed proof is given in our technical report [25].

### *D. Proactive Online algorithm*

Our proactive online algorithm $VPCM$ is shown as Alg. 1. Note that $VPCM$ employs the FTRL method to predict the flow rate $\alpha_i(t+1)$ that minimizes $h_{it}(\alpha_i(t+1))$ in each time slot $t$. With $\alpha_i(t+1)$ obtained, the algorithm then computes the total processing capacity $C_n(t+1)$ and the new instances $x_n^{new}(t+1)$ that need to be launched, and derive the capacities of these new instances, *i.e.*, $c_n^l(t+1)$, or $c_n^{max}$ and $c_n^g(t+1)$, for each VNF $n$. After that, it uses two other algorithms to deploy these newly launched VNF instances and route traffic for service chains with overloaded VNFs, respectively.

Specially, we enqueue the instances that are not used in last time slot into respective buffer queue for each VNF $n$, starting from time slot $t = 2$ (line 3). In each $t$, we first observe the actual flow rates (line 5). If the actual capacity is less than the estimated capacity, we will renew instances popped out from the buffer queue and put them into use with sufficient capacity so that unserved flows can be absorbed (line 7-8). We then compute the residual space capacity and bandwidth capacity for each node and link (line 10).

Next, we explain how to detect the overloaded VNFs in a service chain in detail. Existing literature, like [26], performs scaling trials by adding a new instance for all VNFs in the chain one by one at a time, which is inefficient and resource-consuming. In our case, with predicted flow rate $\alpha_i(t+1)$, the processing capacity $c_n(t+1) = \alpha_i(t+1)\tau_n/L_p$ requested in next time slot $t+1$ can then be estimated, for each VNF $n$ in the chain $i$ (*i.e.*, $\theta_n^i = 1$). By comparing it with current capacity $c_n(t)$ of VNF $n$ in the chain, whether VNF $n$ will be overloaded ($c_n(t+1) > c_n(t)$) or not ($c_n(t+1) \leq c_n(t)$) can be roughly determined. Though it is not necessarily true, we use the results to guide new VNF instance deployment.

Based on this detection method, we call a subprogram **SCR** to use renewed instances from the queues to serve those overloaded VNFs in the chain (line 11), while updating the residual bandwidth capacity (line 12). Then, we predict the flow rates $\alpha_i(t+1)$ with the solution of (12), which introduces a regret of $\frac{3\sqrt{T}-1}{2}\alpha_i^{max}$ in demand prediction over all time

**Algorithm 1:** Proactive Online VNF Provisioning Algorithm for Cost Minimization - *VPCM*

```
   Input: $I$, $N$, $\mathbf{C}$, $\mathbf{B}$, $\theta$, $\alpha^{max}$, $\mathbf{c^{max}}$, $\tau$, $L_p$, $\kappa$
   Output: $\mathbf{c^l}$, $\mathbf{c^g}$, $\mathbf{x^{new}}$, $\phi$, $\mathbf{y}$
   Initialize: $\mathbf{x^{new}} = 0$, $\phi = 0$, $\mathbf{y} = 0$, $\alpha = 0$
1  for $t = 1, 2, \cdots, T$ do
2      for $n = 1, 2, \cdots, N$ and $t \geq 2$ do
3          Enqueue type-$n$ VNF instances that are not used into the
             respective buffer queue, and release spare instances last for
             $\kappa$ time slots;
4      end
5      Observe actual flow rates $\alpha_i^*(t)$, $\forall i \in \mathcal{I}$: $t \in [t_i, t_i + \Delta t_i]$;
6      for $n = 1, 2, \cdots, N$ do
7          Compute the total processing capacity $C_n^*(t)$ requested by
             VNF $n$ and acc. to (1);
8          Renew type-$n$ VNF instances that popped out from the
             logical queue of VNF $n$ with a total capacity of at least
             $C_n^q(t) = \max\{0, C_n^*(t) - C_n(t)\}$ ;
9      end
10     Compute the residual space capacity $C_v(t)$ and residual bandwidth
         capacity $B_e(t)$, $\forall v \in V$, $\forall e \in E$;
11     Call $\mathbf{SCR}(B_e(t), \alpha_i^*(t))$ to absorb unserved flows for service
         chain $i \in \mathcal{I} : t \in [t_i, t_i + \Delta t_i]$ with overloaded VNFs;
12     Update residual bandwidth capacity $B_e(t)$, $\forall e \in E$;
13     Derive predicted $\alpha_i(t+1)$ by solving problem (12), $\forall i \in \mathcal{I}$:
         $t + 1 \in [t_i, t_i + \Delta t_i]$;
14     Add initial demands $\alpha_i(t+1)$, $\forall i : t + 1 = t_i$;
15     for $n = 1, 2, \cdots, N$ do
16         Compute $C_n(t+1)$ acc. to (1) using $\alpha_i(t+1)$ and
             $x_n^{new}(t+1)$ acc. to (2) using $C_n^*(t)$, obtain $c_n^l(t+1)$, or
             $c_n^{max}$ and $c_n^g(t+1)$, $\forall i \in \mathcal{I}$: $t + 1 \in [t_i, t_i + \Delta t_i]$ ;
17         if $x_n^{max}(t+1) > 0$ then
18             Call
                 $\mathbf{NIA}(x_n^{new}(t+1), c_n^l(t+1), c_n^{max}, c_n^g(t+1), C_v(t))$
                 to deploy new type-$n$ VNF instances;
19             Update residual space capacity $C_v(t)$, $\forall v \in V$;
20         end
21     end
22     Call $\mathbf{SCR}(B_e(t), \alpha_i(t+1))$ to set routing path for service chain
         $\forall i \in \mathcal{I} : t + 1 \in [t_i, t_i + \Delta t_i]$ with overloaded VNFs in $t + 1$;
23     Update residual bandwidth capacity $B_e(t)$, $\forall e \in E$;
24 end
```

slots. Added with initial demands starting at $t+1$ (line 14), we can compute the number of new instances that need to be launched and their respective capacities (line 16). Then we call the other subprogram **NIA** to deploy new instances on available nodes, and call **SCR** again to route flows for service chains that have overloaded VNFs. **SCR** and **NIA** will be given in detail in Sec. V, respectively.

## V. Online Algorithms for New Instances Assignment and Flow Routing

We now present the two subprograms **NIA** and **SCR**. Given the number of new type-$n$ VNF instances in need and their respective processing capacities, problems still need to be solved are how to deploy these new VNF instances and route flows of service chain demands to them.

The algorithm for new instances assignment can be viewed as a variant of bin packing problem called *variable-sized bin packing*, where each instance is an item and each server node is a bin with different volumes, *i.e.*, residual space capacities. The problem is NP-hard as it can be viewed as a reduction from the classic bin packing problem. The rational of algorithm design is to minimize the sum of volumes of the nodes used, leaving other nodes more residual space capacities for the assignment of other new VNF instances. The algorithm for service chain routing involves only new arrival packets in flows, allowing for the state migration problem in NFV. The rational of the primal dual algorithm design is to route the arriving flows along alternative admissible paths that contain least utilized links, for purpose of balanced use of bandwidth on links.

### *A. Online Algorithm for Assignment of New Instances*

The algorithm **NIA** in Alg. 2 is based on the best-fit first decreasing loading heuristic. In our case, the "best-fit" means the used node with the maximum residual space capacity after instances assigned. **NIA** first sorts the nodes according to non-decreasing order of their residual space capacity $C_v(t)$, $\forall v \in V$. If $x_n^{new}(t+1) = 1$, it means that only one instance with capacity $c_n^l(t+1)$ needs to be assigned. Based on algorithmic principle, we just need to assign this instance to the first node of the ordered list (line 2-4). Else, $x_n^{new}(t+1) - 1$ instance with capacity $c_n^{max}$ and one instance with capacity $c_n^g(t+1)$ need to be assigned (line 6). For each arrival of new type-$n$ VNF instance, we first attempt to assign it to the "best-fit" already-used node (line 7-9). If this attempt fails, a new node is used and the instance will be assigned to it (line 10-12). Note that $V_n$ denotes the set of nodes used for type-$n$ VNF instances by the algorithm in $t+1$.

Best-fit decreasing heuristic often yields good solutions. However, when the last instances are considered, a new node with capacity that is much larger than that of those last instances might be used. Furthermore, no more instances have the chance to be assigned to this "best-fit" node. To avoid this, we iteratively swap some already-used nodes with unused nodes in $U \setminus V_n$ to improve the solution in a time slot interval, since $x_n^{new}(t+1)$ is finite. Specifically, if an unused node $w$ has enough capacity to hold the instances assigned to a used node $u$ before, and its space capacity is smaller than the used one ($C_w < C_u$), then $w$ will replace $u$ to hold these instances (line 17-19). The actual assignment process is performed at last (line 22).

### *B. Online Algorithm for Service Chain Routing*

After new VNF instances have been assigned, we then need to route flows for service chain demands with overloaded VNFs. To avoid complex state migration problem, we assume that we just route new coming packets in a flow ("new flow") to those new instances in $t+1$, based on the context information provided by FlowTags [27], namely, the routing of other packets in the flow keeps unchanged.

We assume that the set of flow demands with overloaded VNFs is denoted as $\mathcal{I}_f$. Recall that each demand $i \in \mathcal{I}_f$ has a source-destination pair $(o_i, s_i)$. If the overloaded VNFs in $i$ are $n_i^1, n_i^2, \cdots, n_i^d$ and they also follow their order in the chain, we assume the set of admissible paths connecting any node pair $(o_i, s_i)$ is

$$\mathcal{P}(i) = \{(o_i, n_i^1, \cdots, n_i^d, s_i) : n_i^j \in \mathcal{N}, \forall j = 1, \cdots, d\} \quad (18)$$

**Algorithm 2:** Online Algorithm for New Instance Assignment -**NIA**

```
Input: $x_n^{new}(t+1), c_n^l(t+1), c_n^{max}, c_n^g(t+1), C_v(t)$
Initialize: $V_n = \emptyset$, $\phi = 0$
Sort the nodes in $U$ acc. to non-decreasing order $C_v(t)$, $\forall v \in V$;
if $x_n^{new}(t+1) = 1$ then
    $V_n = V_n \cup \{u\}$, where $u$ is the first node in the ordered list $U$, and set $\phi_n^u = 1$;
    Assign this instance in $u$ and return $\phi$;
end
foreach arrival of new type-$n$ VNF instance do
    if instance $n$ can be accommodated into a node in $V_n$ then
        Find the best-fit node $u \in V_n$ for $n$ and set $\phi_n^u = 1$;
    end
    else
        $V_n = V_n \cup \{u\}$, where $u$ is the first node in the ordered list $U$, and set $\phi_n^u = 1$;
    end
end
foreach $u \in V_n$ do
    foreach $w \in U \setminus V_n$ do
        $C_u^n = \sum_{\text{instance } n \text{ will be assgined in } u} c_n$;
        if $C_w \geq C_u^n$ and $C_w < C_u$ then
            $V_n = V_n \setminus \{u\} \cup \{w\}$, and set $\phi_n^u = 0$ while $\phi_n^w = 1$ for each instance $n$ assigned to $u$ before;
        end
    end
end
Execute actual instance assignment in nodes acc. to $\phi$.
```

where each $(o_i, n_i^1, \cdots, n_i^d, s_i)$ denotes the concatenating paths $P_{o_i n_i^1}, P_{n_i^1 n_i^2}, \cdots, P_{n_i^d s_i}$.

For each $p \in \mathcal{P}(i)$, we need to decide a splitting ratio $r_{ip}$, which defines the fraction of "new flow" $i$ along the admissible path $p$. A binary $y_{ep}$ indicates whether a link $e \in E$ is in path $p$. Denote by $\alpha_i^{new}$ the "new flow" rate of $i \in \mathcal{I}_f$ arriving in $t+1$. Then the constraints for the "new flow" routing optimization can be formulated as follows:

$$\sum_{p \in \mathcal{P}_i} r_{ip} \leq 1, \ \forall i \in \mathcal{I}_f \tag{19}$$

$$\sum_{i \in \mathcal{I}_f} \sum_{p \in \mathcal{P}_i} r_{ip} y_{ep} \alpha_i^{new} \leq B_e(t), \ \forall e \in E \tag{20}$$

$$r_{ip} \geq 0, y_{ep} \in \{0, 1\}, \ \forall i \in \mathcal{I}_f, p \in \mathcal{P}(i), e \in E \tag{21}$$

Note that constraint (20) is equivalent to (6) in problem (4), since $y_i(e)$ in (6) can be naturally obtained once $r_{ip}$ is determined, *i.e.*, $r_{ip} y_{ep} \rightarrow y_i(e)$, while $y_{ep}$ is a known binary. The objective can be throughput maximization or other ones, according to the routing performance that the NFV provider pursues. Here we focus on the throughput maximization. Clearly the above problem is a linear programming (LP).

To solve the LP routing problem efficiently, we adopt a primal dual algorithm, which is simple to implement and runs much faster than a general LP solver. Specifically in **SCR**, each admissible path $p \in \mathcal{P}(i)$ is associated with a "path cost", $S_p = \sum_{\{i \in \mathcal{I}_f, e \in p\}} y_i(e) \lambda(e)$, where $\lambda(e)$ is a dual variable. The constant $L^* = \max_{\{i \in \mathcal{I}_f, e \in p\}} y_i(e)$, which denotes the maximum resource usage of each link over all possible paths for the "new flows". $\mu(i)$ is another dual variable and $|E|$ is the number of links. For any $\epsilon > 0$ that is moderately small, **SCR** is given in Alg. 3.

**Algorithm 3:** Online Algorithm for Service Chain Routing -**SCR**

```
Input: $B_e(t), \alpha_i(t+1)$
Initialize: $\lambda(e) = 0, \forall e \in E,\ \mu(i) = 0, \forall i \in \mathcal{I}_f$
$\varphi = L^*/\epsilon$;
Derive $\alpha_i^{new}$ based on $\alpha_i(t+1)$;
foreach arrival of "new flow" demand $i$ do
    $p^* = \arg\min_{p \in \mathcal{P}_i} S_p$;
    if $S_{p^*} < 1$ then
        Route $i$ through $p^*$;
        $\mu(i) = \alpha_i^{new}(1 - S_{p^*})$;
        foreach $e \in p^*$ do
            $\lambda(e) = \lambda(e)\left[1 + \frac{\alpha_i^{new} y_i(e)}{B_e(t)}\right] + \frac{\alpha_i^{new} y_i(e)}{|E| \varphi B_e(t)}$
        end
    end
end
```

### *C. Performance Analysis for Assignment and Routing Algorithms*

We next analyze the performance ratio of the two algorithms, **NIA** and **SCR**, respectively. Let $C_n(OPT)$ denote the minimum total residual capacity of nodes used by optimal algorithm $OPT$, and $C_n(NIA)$ denote the total residual capacity of nodes used by **NIA**, for holding all requested type-$n$ new instances. Then we have the following lemma:

**Lemma 2.** *$C_n(NIA) < \frac{3}{2} C_n(OPT)$, for pre-assigning requested type-$n$ instances in $t+1$ on all available nodes with residual capacity $C_v(t)$, $\forall n \in \mathcal{N}, v \in V$.*

The proof can be derived just as a similar procedure to show the ratio in [20] and therefore is omitted.

**Lemma 3.** *For any moderately small $\epsilon > 0$,* **SCR** *in Alg. 3 is $(1+\epsilon)$-competitive in routing the "new flows", as compared with the primal LP given by (19)-(21).*

Detailed proof is given in our technical report [25].

### *D. Overall Performance Analysis for VPCM*

*VPCM* first predicts the flow rates of service chain demands. Based on the prediction results, it then launches and assigns new requested instances. Recall that the cost incurred in *VPCM* consists of inaccurate VNF provisioning and VNF deployment. Let $\text{Ratio}^T(VPCM)$ denote the competitive ratio of $VPCM$. $P[VPCM]$ and $P[VPCM(z_i^*)]$ are predicted costs incurred by $VPCM$ that uses prediction results and the best predictor $z_i^*$, respectively. $D[NIA(z_i^*)]$ and $D[OPT(z_i^*)]$ are the deployment cost incurred by NIA and the optimal algorithm that use $z_i^*$ to deploy the new VNF instances on available nodes. We have

$$\text{Ratio}^T(VPCM) = \frac{P[VPCM]}{P[VPCM(z_i^*)]} \frac{D[NIA(z_i^*)]}{D[OPT(z_i^*)]}. \tag{22}$$

**Theorem 2.** *The competitive ratio of VPCM in Alg. 1 is upper bounded by $\frac{3}{2}\{1 + \frac{\frac{3}{2}\sum_{i \in \mathcal{I}} s^{max} \alpha_i^{max} L_p}{\sqrt{T} \sum_{n,i} u^{min} \theta_n^i (\alpha_i^+ - z_i^*) \tau_n}\}$, where $s^{max} = \max_{t \in \mathcal{T}} s(t)$, $u^{min} = \min_{t \in \mathcal{T}} u(t)$, and $\alpha_i^+$ is minimum predicted flow rate and $\alpha_i^+ > z_i^*$.*

Detailed proof is given in our technical report [25].

**Remark:** A larger $\frac{s^{max}}{u^{min}}$ indicates a higher cost of revenue loss compared to resource wastage, which means inaccurate prediction has greater influence on the performance ratio of $VPCM$. A larger $L_p$ and smaller $\tau(n)$ leads to smaller lower bound of $P[VPCM(z_i^*)]$, thus larger competitive ratio. As $T$ grows, the influence of inaccurate prediction fades, and $\mathrm{Ratio}^T(VPCM)$ approaches $\frac{3}{2}$, which conforms to the competitive ratio of VNF deployment given in Lemma 2.

Due to space limitation, we provide the complexity analysis of the proposed algorithms in our technical report [25].

## VI. Performance Evaluation

### A. Simulation Setup

We simulate an NFV cloud marketplace that spans over thousands of 5-minute time slots. The service chain demands are derived from .pcap files of the Skype traces [28]. Each record in the files contains a timestamp, a source and destination IP, the captured network traffic and other information from real users. In each time slot $t$, we regard those records that have the same source and destination IP and fall into slot $t$ as a flow. The flow rate is calculated by dividing the total traffic size of the records in the flow by the interval of a time slot. To reflect the traffic fluctuations of a flow over different time slots, we further consider the flows with the same source and destination IP to be the same flow over time. Each flow goes through a service chain containing VNFs randomly selected from Table I. The processing time and capacity of these VNFs conforms to commercial appliances as stated in [9], [5]. We also assume that the packet length is 1024 bytes on average.

TABLE I
Off-the-shelf VNFs

| VNF types<br>Parameters | FW | IDS | IPSec | WAN-opt. |
|---|---|---|---|---|
| Processing time ($\mu$s) | 120 | 160 | 82.76 | 200 |
| Maximum capacity (Mbps) | 400 | 600 | 580 | 100 |

We consider the same topology reported in [9], which is composed of 64 servers. In the basic configuration, 10 Gbps links are used. The space capacity is initialized as 4800 Mbps for each server node to hold VNF instances. For default settings, $s(t)$ and $u(t)$ are randomly set within $[0.1, 1]$ and $0.045 \times [0.1, 1]$, respectively. $d_n(t) = 4s(t)$.

### B. Effectiveness of Demand Prediction

We first examine whether inaccurate estimation of $\alpha_i^{max}$ based on prior experience will lead to high regret. To achieve this, we enlarge (narrow) each $\alpha_i(t)$ by 2 to 5 times to obtain a large (small) $\alpha_i^{max}$. Fig. 1 plots the regret of FTRL when running Alg. 1, with a normal $\alpha_i^{max}$, a larger and a smaller $\alpha_i^{max}$, respectively. For comparison, we also plot the theoretical upper bound in the figure. We can find that with the increase of $t$, all regrets are always much smaller than the theoretical bound. Further, either regret of large or small $\alpha^{max}$ keeps closer and is just a little more than the normal one.

Next we examine the impact of an appropriate selection of learning rate on regret. Fig. 2 shows the difference between two regrets, one with a learning rate of $\eta_t = \frac{\alpha_i^{max}}{\sqrt{t}}$ and the other with a global learning rate of $\eta = \frac{\alpha_i^{max}}{\sqrt{T}}$. We can observe that the gap between the two regrets becomes larger with the increase of $t$, which indicates that FTRL tends to achieve a lower regret.

### C. Impact of Parameters on Ratio

Note that $\mathrm{Ratio}^T(VPCM)$ is computed by (22). We now evaluate the impact of parameters on the ratio. Based on the complete ratio of $VPCM$ provided by Theorem. 2, we first change the ratio of $\frac{s^{max}}{u^{min}}$. A decreasing trend of ratio can be observed in Fig. 3 with the increase of $T$ and in contrast, a increasing trend can be observed with the increase of $\frac{s^{max}}{u^{min}}$. This conforms to the result in Theorem. 2 since a larger $\frac{s^{max}}{u^{min}}$ leads to large ratio with the same $T$. When $\frac{s^{max}}{u^{min}}$ is relatively small, the impact on ratio is not significant, and then $\sqrt{T}$ is the dominant impact.

To evaluate the impact of processing time of VNFs, we assume all service chains just go through one type of VNF, and vary the processing time of this VNF. Fig. 4 also shows the ratio decreases when $T$ grows. While in each $T$, we can observe that the ratio tends to increase when the processing time increases from 80 $\mu$s to 200 $\mu$s. When $T$ is relatively small, the increasing range is notable. When $T = 400$, the impact of $\tau(n)$ is minor since $\sqrt{T}$ is the dominant impact at that moment. This also conforms to Theorem. 2.

### D. Comparison with Other Approaches

We also compare the FTRL approach employed in $VPCM$ with other online learning algorithms, online gradient decent (OGD) [19] and Follow-The-Leader (FTL) [19] without adding a regularizer. We plot the ratio using different algorithms in Fig. 5, with the theoretical upper bound in Theorem. 2 omitted. The results show that $VPCM$ outperforms both OGD and FTL. Next, we implement two other heuristics called constant capacity allocation (CCA) and maximum capacity allocation (MCA). CCA launches new instances with the same processing capacity, which is a constant value below the maximum, and MCA launches new instances all with maximum capacity, regardless of capacity requested. Then, both CCA and MCA use **SCR** and **NIA** to assign these instances and route flows to these instances, respectively. Fig. 6 confirms that $VPCM$ achieves lower overall cost compared with CCA and MCA. As $T$ grows, the cost gap becomes larger.

## VII. Conclusion

In this paper, we study the proactive VNF provisioning problem for NFV providers, considering the fluctuations of traffic traveling service chains. We formulate the problem that minimizes the cost incurred by inaccurate prediction and VNF deployment. We first employ an online learning method which aims to minimize the prediction error, to predict the upcoming traffic flows. When launching new instances based on the prediction outcomes, an adaptive scaling strategy is adopted for saving resources and decreasing deployment cost. We then propose two algorithms that are called by the complete online

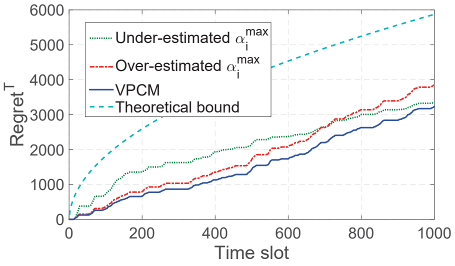

Fig. 1. Influence of inaccurate estimated $\alpha_i^{max}$ on regret of $VPCM$.

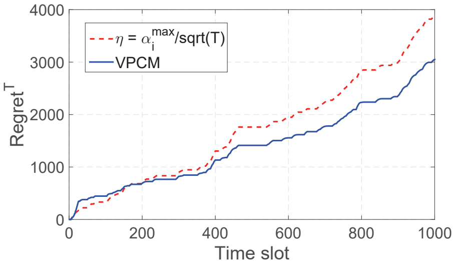

Fig. 2. Regret of $VPCM$ with different learning rate $\eta$.

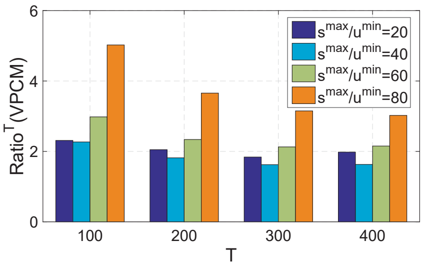

Fig. 3. Ratio of $VPCM$ by varying $s^{max}/u^{min}$.

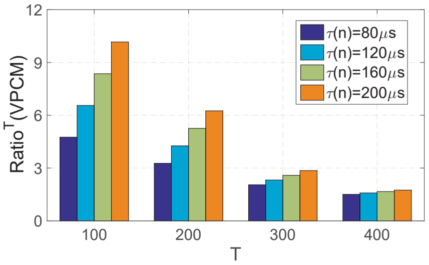

Fig. 4. Ratio of $VPCM$ by varying $\tau(n)$.

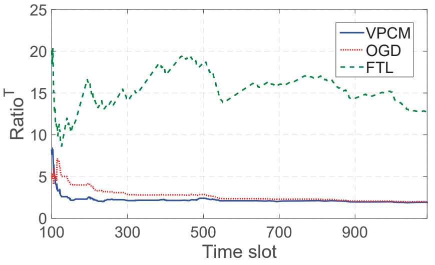

Fig. 5. Ratio of different prediction strategies.

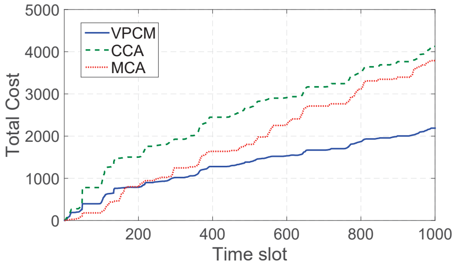

Fig. 6. Overall cost under different approaches.

algorithm, for new instance assignment and service chain routing. Regret analysis for demand prediction and competitive ratios of the two algorithms, along with the complete online algorithm, are provided, which are confirmed by the trace-driven simulation.

## References

[1] "Network Functions Virtualization," https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper3.pdf.

[2] "Service Function Chaining Architecture," https://tools.ietf.org/pdf/rfc7665.pdf.

[3] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 13–24, Aug. 2012.

[4] "NEC/Netcracker NaaS," https://www.netcracker.com/naas.

[5] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic virtual network function placement," in *Proc. of IEEE CloudNet*, Oct 2015, pp. 255–260.

[6] 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 *Proc. of IEEE INFOCOM*, 2017.

[7] R. Mijumbi, S. Hasija, S. Davy, A. Davy, B. Jennings, and R. Boutaba, "A connectionist approach to dynamic resource management for virtualised network functions," in *Proc. of IEEE CNSM*, Oct 2016, pp. 1–9.

[8] T. Wang, H. Xu, and F. Liu, "Multi-resource load balancing for virtual network functions," in *Proc. of IEEE ICDCS*, June 2017, pp. 1322–1332.

[9] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca, "An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures," *IEEE/ACM Transactions on Networking*, vol. PP, no. 99, pp. 1–18, 2017.

[10] X. Wang, C. Wu, F. Le, A. Liu, Z. Li, and F. Lau, "Online VNF Scaling in Datacenters," in *Proc. of IEEE CLOUD*, June 2016, pp. 140–147.

[11] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," *ACM SIGCOMM*, vol. 45, no. 4, pp. 123–137, Aug. 2015.

[12] "Resource Management in Service Chaining," https://tools.ietf.org/pdf/draft-irtf-nfvrg-resource-management-service-chain-03.pdf.

[13] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/merge: System support for elastic execution in virtual middleboxes," in *Proc. of USENIX NSDI*, 2013, pp. 227–240.

[14] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization," in *Proc. of IEEE CloudNet*, Oct 2015, pp. 171–177.

[15] 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. of IEEE ICDCS*, June 2017, pp. 731–741.

[16] X. Fei, F. Liu, H. Xu, and H. Jin, "Towards load-balanced vnf assignment in geo-distributed nfv infrastructure," in *Proc. of IEEE/ACM IWQoS*, June 2017, pp. 1–10.

[17] T. W. Kuo, B. H. Liou, K. C. J. Lin, and M. J. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," in *Proc. of IEEE INFOCOM*, April 2016, pp. 1–9.

[18] H. B. McMahan, "A unified view of regularized dual averaging and mirror descent with implicit updates," *arXiv preprint arXiv:1009.3240*, 2010.

[19] S. Shalev-Shwartz, "Online learning and online convex optimization," *Found. Trends Mach. Learn.*, vol. 4, no. 2, pp. 107–194, Feb. 2012.

[20] D. K. Friesen and M. A. Langston, "Variable sized bin packing," *SIAM Journal on Computing*, vol. 15, no. 1, pp. 222–230, 1986.

[21] G. Karakostas, "Faster approximation schemes for fractional multicommodity flow problems," in *ACM-SIAM SODA*, 2002, pp. 166–173.

[22] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou, "Traffic aware placement of interdependent nfv middleboxes," in *Proc. of IEEE INFOCOM*, 2017.

[23] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos, and J. Kubica, "Ad click prediction: a view from the trenches," in *Proc. of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2013.

[24] N. Chen, A. Agarwal, A. Wierman, S. Barman, and L. L. Andrew, "Online convex optimization using predictions," in *Proc. of ACM SIGMETRICS*, 2015, pp. 191–204.

[25] "Technical Report." [Online]. Available: https://1drv.ms/b/s!Ajku4WFVs-m3cRykraJSCLYy3Ng

[26] A. Gember, R. Grandl, A. Anand, T. Benson, and A. Akella, "Stratos: Virtual middleboxes as first-class entities," *UW-Madison TR1771*, p. 15, 2012.

[27] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *Proc. of USENIX NSDI*, 2014, pp. 543–546.

[28] "TCP STatistic and Analysis Tool: Skype Traces," http://tstat.polito.it/traces-skype.shtml.