Proactive VNF Provisioning with Multi-timescale Cloud Resources: Fusing Online Learning and Online Optimization

Xiaoxi Zhang[†], Chuan Wu[†], Zongpeng Li[§], Francis C.M. Lau[†]
[†]Department of Computer Science, The University of Hong Kong, Email: {xxzhang2,cwu,fcmlau}@cs.hku.hk
[§]Department of Computer Science, University of Calgary, Canada, Email: zongpeng@ucalgary.ca

***Abstract*—Network Function Virtualization (NFV) represents a new paradigm of network service provisioning. NFV providers acquire cloud resources, install virtual network functions (VNFs), assemble VNF service chains for customer usage, and dynamically scale VNF deployment against input traffic fluctuations. While existing literature on VNF scaling mostly adopts a reactive approach, we target a proactive approach that is more practical given the time overhead for VNF deployment. We aim to effectively estimate upcoming traffic rates and adjust VNF deployment *a priori*, for flow service quality assurance and resource cost minimization. We adapt online learning techniques for predicting future service chain workloads. We further combine the online learning method with a multi-timescale online optimization algorithm for VNF scaling, through minimization of the *regret* due to inaccurate demand prediction and minimization of the *cost* incurred by sub-optimal online decisions in a joint online optimization framework. The resulting proactive online VNF provisioning algorithm achieves a good performance guarantee, as shown by both theoretical analysis and simulation under realistic settings.**

I. Introcution

Network Function Virtualization (NFV) promises to ease network service provisioning and scaling at significantly reduced costs, by implementing network functions (middleboxes) as software on standard virtualized platforms (*e.g.*, virtual machines (VMs) in datacenters) instead of dedicated hardware [1]. In contrast with manual hardware middlebox installation, the deployment of *virtualized network functions (VNFs)* is more flexible, by launching a VM instance with a pre-built VNF image. Network functions are typically sequenced into *service chains* to provide practical network service [2], *e.g.*, the service chain "Firewall→IDS→Proxy" is commonly deployed between external users and an enterprise's internal network for access control.

Enterprises and businesses start resorting to VNFs for provisioning network services [3]. Recent initiatives suggest establishing (dedicated) NFV service providers, who own or rent cloud resources, create VNF instances, assemble VNF service chains and offer them to customers on demand, eliminating the need to build and maintain these functions at customers' premises. A European FP7 ICT project T-NOVA [4] introduces an enabling framework for NFV providers to offer VNFs to customers. Customers browse the VNF catalog at the NFV provider, and make *a la carte* orderings of service chains. The emerging NFV marketplace has recently witnessed its vanguard group of providers and vendors [5][6] , racing to lead in the multi-billion dollar market envisioned [7].

At the core of NFV service provisioning is optimal resource scheduling, for efficient and dynamic provisioning of VNF instances that best serves fluctuating network traffic traversing service chains. A delay is typically incurred for VNF deployment, in copying VM images and launching VMs/VNFs. A proactive provisioning approach would thus be much appreciated, adjusting VNF deployment prior to actual traffic arrival based on effective customer demand prediction. However, most existing proposals on dynamic VNF provisioning (*e.g.*, [8][9]) have adopted a reactive approach, adapting VNF deployment to hitherto arrivals, jeopardizing service quality during the adjustment period.

We study a new NFV brokerage service, where a NFV provider predicts upcoming traffic demand, reserves VMs at a public cloud, deploys VNFs and assembles service chains to serve customers over time. On today's public clouds [10][11], VM instances of different resource composition can be reserved for desired durations, *e.g.*, on-demand instances for one-hour and reserved instances for 1 or 3 years at Amazon EC2. A critical challenge arises on which types of VMs the NFV provider should purchase for VNF deployment, based on predicted fluctuation of customer traffic demand. The decisions are to be made promptly upon predicted changes, at a short decision interval on the order of seconds or minutes. VM lifetimes are typically much larger, *e.g.*, hours or months/years in commercial cloud platforms [10][11]. Therefore, VM purchase decisions at one time influence future resource availability and hence future VM purchase decisions. A carefully designed proactive online algorithm should (i) effectively predict upcoming traffic demand to expedite service chain provisioning, and (ii) strategically and proactively reserve VMs and deploy VNFs based on the predicted upcoming demand as well as a future perspective, to best serve customer traffic with minimal overall cost over the long run.

Algorithm design for the above goals faces a number of technical challenges. Given the multiple time scales of VM duration, online VM purchase decisions are coupled with future decisions at multiple time scales. Moreover, the overall VM purchase and deployment cost is a non-convex objective function, further escalating the challenge. In online (convex) optimization problems solved by online learning techniques, decisions at each time are made based on predicted (learned) input for the current time, without direct coupling with future decisions [12][13]. Exceptions [14][15] study online learning with decisions coupled over time in a convex function, rely-

ing on lookahead windows. In our problem, online demand prediction in a smaller time scale is closely related with online optimization in longer time scales, in both constraints and the objective function. Moreover, we do not assume a lookahead window that provides future information. How to design an online algorithm fusing online learning and online optimization, achieving provable performance bounds, is an intriguing task of practical relevance.

We design an online, proactive VNF provisioning algorithm for the novel NFV brokerage service, addressing the above challenges. We model proactive VNF provisioning into an online optimization framework that jointly addresses regret minimization for demand learning and long-term VM cost minimization. The algorithm design consists of two main modules: (i) an online learning approach to predict traffic demand along the service chains, and (ii) an online algorithm for purchasing VMs of different durations and deploying VNFs on VMs according to the predicted demand, simultaneously retaining a future perspective for long-term cost optimality.

In the learning module, we exploit an efficient online gradient descent method to predict upcoming traffic demand along the VNF chains, minimizing the loss due to prediction errors over time. We carefully design a randomized convexification technique to convert a non-convex loss minimization into an online convex optimization problem, such that the gradient descent method can be effectively applied. An expected regret bound sub-linear to the time horizon is proven, as compared to the best static prediction strategy (a common benchmark for analyzing online learning techniques).

The predicted traffic demand is fed as input to the online algorithm module for VM purchase and VNF deployment. We exploit the break-even principle from ski-rental algorithms in our online algorithm design, and apply it in multiple levels of online decision making, for purchasing VMs of different durations. Assuming a given traffic demand sequence, the online algorithm module achieves a constant competitive ratio of 3, as compared to the offline optimum derived under the same demand sequence.

The online learning module and the online optimization module are combined into our complete proactive online VNF provisioning algorithm, $POLAR$. A new performance ratio is constructed to evaluate $POLAR$, which jointly measures the regret in demand prediction and the competitiveness of online VM purchase. The ratio compares the expected overall cost incurred by $POLAR$ with that by a strategy using the best static demand predictor and the optimal strategy for VM purchase with full knowledge of demand over the system lifetime. The ratio is upper bounded arbitrarily closely by the constant 3, when the system lifetime goes to infinity. Our simulation study under realistic settings further demonstrates that $POLAR$ outperforms alternative approaches in both demand prediction and overall cost minimization.

II. Related Work

A few recent studies investigated optimal placement and scaling of VNF instances and traffic routing in VNF service chains for cost minimization. VNF-P [16] presents a one-time optimization model and a heuristic algorithm, for VNF placement, considering hybrid deployment where part of the network service is provided by dedicated hardware and part by VNF instances. Cohen *et al.* [17] design approximation algorithms for VNF placement across datacenters, to minimize the distance cost between clients and VNFs and VNF setup costs, and present rigorous analysis.

The above literature deals with one-off placement of the NFV service chains, ignoring the dynamic nature of an NFV system. Wang *et al.* [8] design cost-minimizing online algorithms for dynamic VNF provisioning in the NFV provider's cloud data center. Ghaznavi *et al.* [9] design online algorithms to optimize placement of VNF instances in response to on-demand workload, considering the trade-off between bandwidth and host resource consumption. These dynamic schemes are reactive in nature, scheduling VNF deployment according to arrived demand. In contrast, we target a proactive online algorithm that deploys VNFs according to predicted demand, while guaranteeing a performance bound.

Recently, cloud brokerage service has been investigated. Zheng *et al.* [18] propose the business model where a cloud virtual service provider rents cloud resources from a cloud provider and resells them to users. They study economic issues based on a stochastic job scheduling problem. Wang *et al.* [19] investigate a cloud brokerage service and minimize the cost in choosing different options of resources from a cloud provider to meet users' homogenous demand over time. They further assume that the billing cycle of an instance equals one decision time slot, which simplifies the model but is not practical.

Online learning based algorithms have been proven effective in solving online convex optimization problems [12][13] and some typical non-convex problems, *e.g.,* Prediction with Expert Advice problem [12]. Our problem can not be encoded into a convex problem or reduced to classic problems in online learning settings. Ski-rental problem and its variations have been extensively studied [20]. However, none of them deals with multiple time scales of online resource reservation.

III. Problem Model

A. Types and Durations of VM instances

We consider a dedicated NFV service provider who operates for a large number T of time slots, each on the order of seconds or minutes. The cloud provides M types of VM instances with different compositions of resources such as CPU, RAM, and Disk storage. Each VM instance can be purchased as *short-term instances* or *long-term instances*, as exemplified by the on-demand instances (billed hourly) and reserved instances (reserved for $1-3$ years) on Amazon EC2 [10], respectively. Let τ^s be the billing cycle of a short-term instance ($\tau^s = 60$ if billing cycle is an hour and each time slot is a minute). Let α_m be the unit price of a type-m short-term VM instance per billing cycle, $\forall m \in [M] = \{1, 2, \ldots, M\}$, *e.g.*, on-demand instances on Amazon EC2 have hourly prices within $[\$0.0065, \$13.338]$. Each long-term instance is reserved for τ^l time slots, with a one-time upfront payment of $\mu_m \geq 0$,

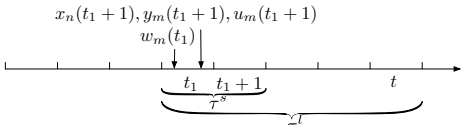

Fig. 1: Time slots, VM durations, and online decisions

and a unit price γ_m per billing cycle of length τ^s in τ^l, for each type-m instance. For example, an EC2 reserved instance `c4.2xlarge` on a 1-year term incurs a \$2303 initial charge, and a significantly reduced hourly fee of \$0.263. The overall price of a long-term instance, β_m, equals $\mu_m+\gamma_m\times\frac{\tau^l}{\tau^s}$, charged no matter whether the VM is in use or not [21]. An illustration of the time slots and VM durations is given in Fig. 1.

B. VNFs and Service Chain Requests

The NFV service provider offers N types of VNFs, *e.g.*, firewall, NAT, IDS. A total number J of service chain requests arrive during $[T]=\{1,2,\ldots,T\}$. A request $j\in[J]=\{1,2,\ldots,J\}$ arriving at t_j specifies an ordered sequence of VNFs composing service chain j, start time t_j^- and end time t_j^+ for using this service chain, where $t_j<t_j^-\leq t_j^+$. For example, an enterprise outsourcing its access service may request for the service chain *Firewall→IDS→Proxy*. Let $\mathbb{L}^j$ be the set of VNFs in service chain j. We use n^- and n^+ to represent the successor VNF and the predecessor VNF of a type-n VNF in the chain, respectively.

Let $\pi_j(t)$ be the total traffic rate (*a.k.a. flow rate*) arriving at the first VNF in service chain j in t. $\pi_j(t)=0$ for $t\in[0,t_j^-)\cup(t_j^+,\infty)$. The flow rate may vary at different hops along the VNF chain: tunneling gateway VNFs (*e.g.*, IPSec/SSL VPN and media gateways) may increase (decrease) the packet size for encapsulation (decapsulation) [22]; security VNFs (*e.g.*, firewall, IDS) drop packets that violate security policies [23]. Let $\lambda_n^j(t)$ be the flow rate change ratio of service chain j at VNF n in t, such that the outgoing rate of flow j (aggregate flow of service chain j) after traversing an instance of VNF n is on average $\lambda_n^j(t)$ times the incoming rate to the n. The arrival flow rate to a service chain and the change ratios vary over time and are only known after the traffic has arrived and has been processed. We assume a customer provides a rough estimate (*e.g.*, based on prior experience) of its flow rate for initial service chain deployment.

C. Service Chain Provisioning

The NFV provider provisions a requested service chain by installing specified VNFs on VMs purchased from the cloud. Each type of VNF has one target type of hosting VM, based on preferred hardware configuration for running the VNF. For example, a Bro IDS [24] can be provisioned using a `c4.2large` instance in Amazon EC2, which best fits the recommended configuration of a compute-optimized processor with 8 cores. The VM running a type-n VNF is an *instance of VNF* n, and let C_n denote its flow processing capacity (*e.g.*, in MBps). Multiple instances of VNF n may be needed in a service chain to provide sufficient capacity for flow processing.

Each type of VM can be used for hosting different types of VNFs (if it represents the best fit for the VNF software): in any time slot, only one VNF can be running on one VM, but the same VM can run different VNFs over different time slots. Let $\mathbb{1}_n^m$ indicate whether type-m VMs can host type-n VNFs ($\mathbb{1}_n^m=1$) or not ($\mathbb{1}_n^m=0$), where $\sum_{m\in[M]}\mathbb{1}_n^m=1$ and $\sum_{n\in[N]}\mathbb{1}_n^m\geq 1$. For flow routing along a service chain, the NFV provider informs the customer the IP address(es) of instance(s) of the first VNF in the chain, and installs forwarding rules on VNF instances to ensure that flows belonging to a particular customer are forwarded along the correct chain.

D. Online Decisions and Offline Optimization Problem

The NFV provider deploys service chains in a proactive fashion. In time slot t, the NFV provider predicts the upcoming VNF demand due to fluctuation of flow rate $\pi_j(t+1)$ and rate change ratios along each existing service chain j, $\forall j\in[J], t_j^-\leq t+1\leq t_j^+$. Then together with the customer-provided initial demand of new service chains to start in $t+1$,[1] it checks whether VNFs of sufficient capacity can be provisioned on already purchased VMs (still valid in $t+1$). If the capacity is insufficient, the provider purchases $y_m(t+1)\geq 0$ new type-m long-term VMs and $u_m(t+1)\geq 0$ new type-m short-term VM, to deploy enough VNFs for serving flows in $t+1$.

The NFV provider maintains a small amount of on-premise resources, *e.g.*, using a private server cluster running standby VMs, for running VNFs absorbing unserved flows due to inaccurate demand prediction. Let $s_m(t)$ be the cost of using a type-m backup VM in t, which is much higher than the prices for long-term or short-term VM instances from the public cloud. In t, with flow arrival (*i.e.*, $\pi_j(t)$'s and $\lambda_n^j(t)$'s are revealed), $w_m(t)\geq 0$ type-m backup VMs are used if the pre-deployed instances are insufficient to handle all the flows.

Specifically, the NFV provider makes the following decisions in each time slot t (Fig. 1): (1) the numbers of backup VMs of different types for running VNFs to serve unserved traffic in t (that exceeds the processing capacity of VNFs pre-deployed in $t-1$), *i.e.*, $w_m(t),\forall m\in[M]$; (2) the numbers of different types of VNF instances to deploy in $t+1$, denoted by $x_n(t+1),\forall n\in[N]=\{1,2,\ldots,N\}$; (3) the numbers of new long-term and short-term VMs of different types to purchase for running those VNFs to serve traffic in $t+1$, *i.e.*, $y_m(t+1)$ and $u_m(t+1)$, $\forall m\in[M]$.

The provider aims to minimize the overall cost over the T time slots, consisting of the following components. (i) Total cost of backup VMs: $\sum_{t\in[T]}\sum_{m\in[M]}s_m(t)w_m(t)$. (ii) Total cost of purchasing new long-term and short-term VM instances: $\sum_{t\in[T]}\sum_{m\in[M]}\alpha_m u_m(t)+\beta_m y_m(t)$. (iii) Total cost of deploying VNFs on VM instances, as derived below.

A deployment cost d_m is incurred for copying VM image containing the VNFs to a type-m VM. Since the same VM can be used for running different VNFs at different times, the NFV provider includes all VNF software that a VM can host in the VM image, and only launches a particular VNF

[1]The NFV provider deploys VNFs at time slot $t_j^- - 1$ for new service chains whose start time is t_j^-.

when it decides to enable the VM as an instance of that VNF. Deployment cost is only incurred when starting a new VM, not upon changing VNF deployment on the VM. To cover the need of $u_m(t+1)$ new short-term VMs, the NFV provider maximally exploits the $u_m(t+1-\tau^s)$ short-term VMs purchased at $t+1-\tau^s$ that expire at t, by renewing them for another billing cycle τ^s, avoiding re-copying VNF images. The deployment cost for short-term VM instances at t is incurred only for $(u_m(t+1)-u_m(t+1-\tau^s))^+ = \max\{u_m(t+1)-u_m(t+1-\tau^s), 0\}$ type-m VMs, purchased anew from the cloud. However, long-term VM instances cannot be renewed (*e.g.*, see current practice of reserved instances on Amazon EC2 [25], such that all $y_m(t+1)$ long-term VMs in need should be purchased anew. Therefore, the total cost for deploying VNFs on purchased VM instances is $\sum_{t\in[T]}\sum_{m\in[M]} d_m(y_m(t)+[u_m(t)-u_m(t-\tau^s)]^+)$.

The overall cost minimization problem is:

$$\begin{aligned}\text{minimize} \sum_{t\in[T]}\sum_{m\in[M]} \big\{&\alpha_m u_m(t) + \beta_m y_m(t) \\ &+ d_m\big(y_m(t) + [u_m(t) - u_m(t-\tau^s)]^+\big) + s_m(t)w_m(t)\big\} \quad (1)\end{aligned}$$

subject to:

$$\sum_{n\in[N]} x_n(t)\mathbb{1}_n^m \le \sum_{t-\tau^l+1}^{t} y_m(k) + \sum_{t-\tau^s+1}^{t} u_m(k) + w_m(t), \quad \forall m\in[M], t \quad (1a)$$

$$\sum_{\substack{j\in[J]:\\ t_j^-\le t\le t_j^+, n\in\mathbb{L}^j}} z_{n^-}^n(j,t) \le C_n x_n(t), \quad \forall n\in[N], t \quad (1b)$$

$$z_0^{n_1}(j,t) = \pi_j(t), \quad \forall j\in[J], t \quad (1c)$$

$$z_n^{n^+}(j,t) = \lambda_n^j(t) z_{n^-}^n(j,t), \quad \forall j\in[J], n\in\mathbb{L}^j, t \quad (1d)$$

$$z_{n^-}^n(j,t) \ge 0, \quad \forall j\in[J], n\in\mathbb{L}^j, t \quad (1e)$$

$$w_m(t), y_m(t), u_m(t), x_n(t) \in \mathbb{Z}_+, \quad \forall n\in[N], m\in[M], t \quad (1f)$$

Constraint (1a) ensures that the total number of valid type-m VMs (from both the cloud and the backup pool) is sufficient to serve all VNF instances requiring these VMs. Here $\sum_{k=t-\tau^l+1}^{t} y_m(k)$ and $\sum_{k=t-\tau^s+1}^{t} u_m(k)$ are the numbers of valid long-term and short-term instances purchased during $[t-\tau^l, t-1]$ and $[t-\tau^s, t-1]$, respectively. $z_n^{n^+}(j,t)$ and $z_{n^-}^n(j,t)$ denote the aggregate rates of flows from instances of VNF n to instances of next-hop VNF n^+, and from instances of previous-hop VNF n^- to instances of VNF n, along service chain j in t. Constraint (1b) guarantees that the deployed type-n VNF instances are sufficient to process incoming traffic from all valid service chains using this VNF at t. We allow a VNF instance to serve flows from multiple service chains. Let n_1 be the first VNF in a service chain, and $z_0^{n_1}(j,t)$ be the rate of flows coming into instances of the first VNF in chain j in t. (1c) describes the aggregate incoming traffic rate to each service chain in t. (1d) models the flow rate change by ratio $\lambda_n^j(t)$ when flow j traverses VNF n in t.

Given a solution to the optimization problem, the detailed VNF to VM mapping in t can be decided: $x_n(t)\mathbb{1}_n^m$ VMs of type m out of all the available type-m VMs (RHS of (1a)) are used to provision type-n VNF in t (assigned in the preference order of long-term, short-term and backup VMs of type m). The VNF instances can be easily allocated to different service chains according to their flow rates and processing capacity of each VNF instance, with cross-chain instance sharing permitted. The flows in service chain j, with aggregate rate $z_n^{n^+}(j,t)$, can be routed from instances of VNF n to instances of VNF n^+ proportionally, according to processing capacities allocated to chain j on instances of VNF n^+ [26]. Key notation is summarized in Table I.

TABLE I: Notation

M	total # of VM types	N	total # of VNF types
T	total # of time slots	J	total # of service chains
τ^s (τ^l)	# of time slots of short-term (long-term) VM instance duration		
$y_m(t)$ ($u_m(t)$)	# of long-term (short-term) VM instances of type m purchased at time slot t		
α_m	price per type-m short-term VM instance		
β_m	(upfront fee + total hourly fee over τ^l) per type-m long-term VM instance		
d_m	deployment cost for a type-m VM instance		
$w_m(t)$	# of backup type-m VMs used at t		
$s_m(t)$	price of each backup type-m VM used at t		
$x_n(t)$	# of type-n VNF instances that run at t		
$\mathbb{1}_n^m$	indicator of whether type-n VNFs should be deployed on type-m VMs		
C_n	processing capacity of a type-n VNF instance		
$\pi_j(t)$	total traffic rate to first VNF of chain j at t		
$z_n^{n^+}(j,t)$ ($z_{n^-}^n(j,t)$)	aggregate traffic rate from VNF n (n^-) to next-hop n^+ (n) of chain j at t		
$\lambda_n^j(t)$	rate change ratio by passing VNF n in j at t		

IV. Proactive Online Algorithm for VNF Provisioning

We introduce *POLAR*, a Proactive OnLine AlgoRithm, for the NFV provider to decide $x_n(t+1)$, $y_m(t+1)$ and $u_m(t+1)$, for all $n\in[N]$ and $m\in[M]$, in t.

A. Preliminaries

If we know the incoming flow rates to service chains $\pi_j(t+1)$'s and ratios $\lambda_n^j(t+1)$'s, we can compute $x_n(t+1)$ based on (1b), dividing the total incoming traffic rate to VNF n by processing capacity C_n. By (1c) and (1d), the overall rate of flows arriving at type-n VNF instances in service chain j is $z_{n^-}^n(j,t+1) = \hat{\lambda}_n^j(t+1)\pi_j(t+1)$, where $\hat{\lambda}_n^j(t+1)$ is the cumulative rate change ratio along chain j in $t+1$, before flow j reaches VNF n, computed as follows: ($\forall t\in[T], j\in[J], n\in\mathbb{L}^j$): $\hat{\lambda}_{n_1}^j(t)=1, \hat{\lambda}_n^j(t) = \hat{\lambda}_{n^-}^j(t)\lambda_{n^-}^j(t)$. We thus have

$$x_n(t+1) = \Big\lceil \frac{\sum_{j\in[J]:t_j^-\le t+1\le t_j^+, n\in\mathbb{L}^j} \hat{\lambda}_n^j(t+1)\pi_j(t+1)}{C_n} \Big\rceil \quad (2)$$

Then the total number of type-m VMs needed in $t+1$ to provision all VNFs (LHS of (1a)), denoted by $v_m(t+1)$, is:

$$v_m(t+1) = \sum_{n\in[N]} x_n(t+1)\mathbb{1}_n^m \quad (3)$$

Given $v_m(t+1)$, we can compute $y_m(t+1)$ and $u_m(t+1)$ to ensure that new and previously purchased type-m instances meet the demand in $t+1$, *i.e.*, $\sum_{k=t-\tau^l+2}^{t+1} y_m(k) + \sum_{k=t-\tau^s+2}^{t+1} u_m(k) \ge v_m(t+1)$.

However, in the first place, we do not know $\pi_j(t+1)$ and $\lambda_n^j(t+1)$ at time t. We seek an online learning algorithm *to predict the demand*. Instead of predicting $\pi_j(t+1)$ and $\lambda_n^j(t+1)$, we predict $x_n(t+1)$ directly, based on which we can derive $v_m(t+1)$, $y_m(t+1)$ and $u_m(t+1)$.

B. Minimizing Loss in VNF Demand Prediction

Let $x_n^*(t+1)$ be the actual demand for type-n VNFs revealed in $t+1$, based on actual flow rates. $v_m^*(t+1)$ denotes the actual demand for type-m VMs computed by $x_n^*(t+1)$ using (3). There are three cases for $v_m(t+1)$ computed at t:

(i) $v_m(t+1) = v_m^*(t+1)$. By (1a), we have $w_m(t+1) = \max\{v_m^*(t+1) - (\sum_{k=t-\tau^l+2}^{t+1} y_m(k) + \sum_{k=t-\tau^s+2}^{t+1} u_m(k)), 0\} = 0$, *i.e.*, no backup type-m VMs is needed in $t+1$.

(ii) $v_m(t+1) < v_m^*(t+1)$ (under-provisioning). We may have $w_m(t+1) > 0$. As compared to the objective value of (1) in case (i), cost for purchasing and deploying cloud VMs is smaller using the under-estimation, but backup VM cost $s_m(t)w_m(t)$ is larger, leading to a larger overall cost (since $s_m(t)$ is larger than price of any VM in the cloud).

(iii) $v_m(t+1) > v_m^*(t+1)$ (over-provisioning). We have $w_m(t+1) = 0$ (no backup VM needed), but a larger cloud VM purchase and deployment cost is incurred, as compared to case (i), due to over-provisioning of those VMs.

Loss minimization. We seek to minimize the prediction error $|x_n(t) - x_n^*(t)|$ (*a.k.a.* the loss function), to bound the sub-optimality gap in overall cost due to imperfect prediction. A loss minimization problem is constructed as follows, $\forall n \in [N]$:

$$\text{minimize}_{\mathrm{x_n(t)}\in\mathbb{Z}_+,\forall \mathrm{t}\in[\mathrm{T}]} \quad \sum_{\mathrm{t}\in[\mathrm{T}]} |\mathrm{x_n(t)} - \mathrm{x_n^*(t)}| \tag{4}$$

We solve (4) through an online gradient descent (OGD) method, which derives $x_n(t+1)$ in each t based on past prediction errors. OGD uses the gradient of the objective function at a single step to approximate the best static predictor over all time slots [13]. The variable update step in OGD is simple but proven effective for solving online convex optimization problems. However, (4) is non-convex, due to integrality of $x_n(t)$'s. We convert (4) into a convex problem first, whose objective value equals that in (4) in expectation.

Randomized Convexification. We construct a surrogate loss function $f_{nt}(\cdot)$ using a convex function on variable $\theta_n(t) \geq 0$:

$$f_{nt}(\theta_n(t)) = |\theta_n(t) - x_n^*(t)|, \quad \forall n \in [N], t \in [T], \tag{5}$$

and convert (4) into the following minimization problem:

$$\text{minimize}_{\theta_{\mathrm{n}}(\mathrm{t})\geq 0,\forall \mathrm{t}\in[\mathrm{T}]} \quad \sum_{\mathrm{t}\in[\mathrm{T}]} \mathrm{f_{nt}}(\theta_{\mathrm{n}}(\mathrm{t})) \tag{6}$$

We obtain $x_n(t)$ by randomized rounding of $\theta_n(t)$ to one of its two nearest integers according to the distribution in (7):

$$\mathcal{D}_{x_n(t)}(\theta_n(t)) = \begin{cases} Pr[x_n(t) = \lfloor \theta_n(t) \rfloor + 1] = \theta_n(t) - \lfloor \theta_n(t) \rfloor \\ Pr[x_n(t) = \lfloor \theta_n(t) \rfloor] = 1 - \theta_n(t) + \lfloor \theta_n(t) \rfloor \end{cases} \tag{7}$$

The value of $f_{nt}(\theta_n(t))$ then equals the loss function $|x_n(t) - x_n^*(t)|$ in expectation, as shown in the following Lemma. Detailed proofs of lemmas and theorems can be found in [27].

Algorithm 1: Proactive Online VNF Provisioning Algorithm – $POLAR$

Input: M, N, $\mathbf{C}$, α, β, $\mathbf{d}$, $\mathbf{x}^{*max}$
Output: $\mathbf{x}$, $\mathbf{y}$, $\mathbf{u}$, $\mathbf{w}$
Initialize: $\mathbf{x} = \mathbf{0}$, $\mathbf{y} = \mathbf{0}$, $\hat{\mathbf{y}} = 0$, $\mathbf{u} = \mathbf{0}$, $i = 1$; $\theta = \mathbf{0}$, $\phi = 2$, $\eta = 0$

1 **for** $t = 1, 2, \cdots, T$ **do**
2 Observe actual flow rates $\pi_j^*(t)$ and flow rate change ratios $\lambda_n^{j*}(t)$, $\forall n \in [N]$, $j \in [J] : t_j^- \leq t \leq t_j^+$;
3 Derive actual total number of type-n VNFs needed, $x_n^*(t)$ acc. to (2) using $\pi_j^*(t)$, $\lambda_n^{j*}(t)$, $\forall n \in [N]$;
4 **for** $m = 1, 2, \cdots, M$ **do**
5 Derive actual total number of type-m VMs needed, $v_m^*(t)$ acc. to $x_n^*(t)$ and (3);
6 Deploy backup type-m VMs to absorb flows unserved by cloud-based VMs at $w_m(t) = \max\{v_m^*(t) - (\sum_{k=t-\tau^l+1}^{t} y_m(k) + \sum_{k=t-\tau^s+1}^{t} u_m(k)), 0\}$;
7 **end**
8 **if** $t+1 > 2^i$ **then**
9 $i = i+1$, $\phi = 2^i$;
10 **end**
11 **for** $n = 1, 2, \cdots, N$ **do**
12 $\eta = \frac{x_n^{*max}}{\sqrt{2\phi}}$;
13 $\theta_n(t+1) = \theta_n(t) - \eta \nabla f_{nt}(\theta_n(t))$;
14 **end**
15 Derive prediction $x_n(t+1)$ acc. to (7), $\forall n \in [N]$;
16 **for** $m = 1, 2, \cdots, M$ **do**
17 Compute predicted $v_m(t+1)$ acc. to (3);
18 Add initial demand from new service chains (to start at $t+1$) into $x_n(t+1)$ and $v_m(t+1)$;
19 Call $\mathbf{LTP}(t, \mathbf{v}_m, \hat{\mathbf{y}}_m, \tau^l, \tau^s, \alpha_m, \beta_m, d_m)$ to derive $y_m(t+1)$ and purchase a corresponding number of long-term instances;
20 Call $\mathbf{STP}(t, v_m(t+1), \hat{\mathbf{y}}_m, \mathbf{u}_m, \tau^l, \tau^s, \alpha_m, d_m)$ to derive $u_m(t+1)$ and purchase/renew a corresponding number of short-term instances;
21 **end**
22 **for** $n = 1, 2, \cdots, N$ **do**
23 Launch VNF n on $x_n(t+1)\mathbb{1}_n^m$ type-m VMs, $\forall m \in [M]$;
24 **end**
25 **end**

Lemma 1. *When $x_n(t)$ is drawn from the distribution $\mathcal{D}_{x_n(t)}(\theta_n(t))$ in* (7)*, we have*

$$f_{nt}(\theta_n(t)) = |E[x_n(t)] - x_n^*(t)| = E[|x_n(t) - x_n^*(t)|] \tag{8}$$

C. Proactive Online Algorithm

As shown in Alg. 1, our proactive online algorithm $POLAR$ uses the OGD method to compute $\theta_n(t+1)$ that minimizes $f_{n(t+1)}(\theta_n(t+1))$ in each time slot t. It then produces $x_n(t+1)$ following (7) and $v_m(t+1)$ using (3). Next it uses additional online algorithm modules to derive the numbers of new long-term and short-term instances to purchase for $t+1$ according to $v_m(t+1)$.

Algorithm Procedure. In each t, we observe the actual arrived flow rates and change ratios (line 2). If the flow demands exceed the capacities of pre-deployed VNFs, decide the numbers of backup VMs needed and deploy VNFs on them according to the demand of unserved flows (lines 3-7). Next,

predict future VNF demand: obtaining $\theta_n(t+1)$ using online gradient descent method (line 13) according to $\theta_n(t)$ and: $\nabla f_{nt}(\theta_n(t))$ equals 1 if $\theta_n(t) > x_n^*(t)$; -1 if $\theta_n(t) < x_n^*(t)$; 0, otherwise. Then, we round $\theta_n(t+1)$ to $x_n(t+1)$ (line 15).

Here η is the step size to update $\theta_n(t)$, computed according to line 12. In an OGD method, η is typically set proportional to $\frac{1}{\sqrt{T}}$ (*e.g.*, $\frac{x_n^{*max}}{\sqrt{2T}}$). Not necessarily knowing the system lifespan T during the online process, we use ϕ in η (line 12) instead, which is a forecast of T adjusted over time. This introduces a multiplicative loss factor of $\frac{\sqrt{2}}{\sqrt{2}-1}$ in the regret of our demand prediction, to be analyzed in Theorem 1. x_n^{*max} indicates the upper-bound of $x_n(t)$, such that $x_n(t) \leq x_n^{*max}, \forall t$. While x_n^{*max} may be unknown during the execution of Alg. 1, we can use an estimate of x_n^{*max} (*e.g.*, based on past experience); we evaluate in Sec. VI the consequences of inaccurate estimation.

Given prediction $x_n(t+1)$ for all n, we can obtain $v_m(t+1)$ according to (3) (line 17). We may also boost $v_m(t+1)$ if there are new service chains to start at $t+1$ (line 18). Then call sub-routine **LTP** to decide how many long-term VM instances to purchase (line 19), and call subroutine **STP** to decide the number of short-term VM instances to purchase or renew to serve the remaining demand (line 20). **LTP** and **STP** will be introduced in Alg. 2 and Alg. 3 in Sec. V, respectively.

D. Regret Analysis for VNF Demand Prediction

We analyze performance of our online prediction of $x_n(t+1)$, by computing an expected regret over random choices of $x_n(t+1)$'s. The expected regret of our prediction of type-n VNFs is computed by comparing the overall loss (objective value of (4)) incurred by our algorithm and by the best static prediction strategy [14]. The latter uses the same $\bar{\theta}_n$ (=$E[x_n(t)]$) to produce $x_n(t)$ following the distribution in (7) in all time slots, and achieves the smallest overall loss in expectation among all such static predictors $\bar{\theta}_n \geq 0$. Let $\bar{\theta}'_n$ denote the best static predictor. We have

$$\begin{aligned}\bar{\theta}'_n =& \text{argmin}_{\bar{\theta}_n \geq 0} \sum_{t\in[T]} |\bar{\theta}_n - x_n^*(t)| \\ =& \text{argmin}_{\bar{\theta}_n \geq 0} \sum_{t\in[T]} E_{x_n(t)\sim\mathcal{D}_{x_n(t)}(\bar{\theta}_n)}[|x_n(t) - x_n^*(t)|] \quad (9)\end{aligned}$$

The expected regret is computed as

$$\begin{aligned}&\text{Regret}_n^T(POLAR) = E[\sum_{t\in[T]} |x_n(t) - x_n^*(t)|] \\ &- \min_{\bar{\theta}_n \geq 0} \sum_{t\in[T]} E_{x_n(t)\sim\mathcal{D}_{x_n(t)}(\bar{\theta}_n)}[|x_n(t) - x_n^*(t)|] \quad (10)\end{aligned}$$

The following theorem upper-bounds the overall regret.

Theorem 1. *The regret of $POLAR$ in Alg. 1 in predicting VNF demands, as compared to the best static prediction strategy that uses $\bar{\theta}'_n$ in* (9) *for all $t \in [T]$, is upper-bounded:*

$$\text{Regret}^T(POLAR) = \sum_{n\in[N]} \text{Regret}_n^T \leq \frac{2\sqrt{T}}{\sqrt{2}-1} \sum_{n\in[N]} x_n^{*max}$$

where x_n^{*max} is the upper-bound of $x_n^*(t)$ over all $t \in [T]$.

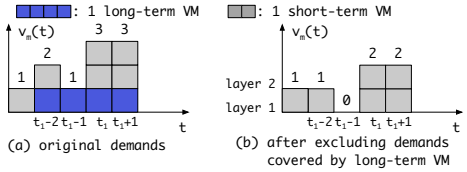

Fig. 2: An illustration of VM demands in Alg. 2

V. Online Algorithm for VM Purchase

Given the predicted total number of type-m VMs in need, $v_m(t+1)$, Alg. 1 uses **LTP** to decide $y_m(t+1)$ long-term VMs to purchase to fulfil $v_m(t+1)$, and then calls **STP** to decide $u_m(t+1)$ short-term VMs to cover the discrepancy. The key rationale is to use cheaper long-term VMs to meet long-lasting demand as much as possible, for cost minimization.

Our algorithm design follows the key principle of break-even algorithms for online ski-rental problems [20], in which a ski is purchased when a threshold of renting days is reached, and the threshold is relevant to the ratio of purchasing cost vs. renting cost. Our online decisions are on reserving long-term VM instances for long use or short-term instances for brief usage. Expired short-term VMs may also be renewed continuously even if they are not in immediate need, to save deployment costs in case that the demand arises soon again.

A. Online Algorithm for Reserving Long-term VMs

LTP in Alg. 2 is designed based on a VM demand grid in Fig. 2 (a). Each block represents the demand of 1 type-m VM in one time slot. Each block can be covered by one long-term or short-term instance. Suppose one long-term (short-term) instance lasts for 4 time slots (2 time slots) in the example in Fig. 2. Intuitively, in each layer, if the cost by using short-term instances to fulfil the demand over $[t+1, t+\tau^l]$ exceeds the cost of a long-term VM ($\beta_m + d_m$), we should use a long-term VM. The critical question is that we do not know the future demand beyond $t+1$. The idea is to use the demand and VM purchase decisions in the past τ^l time slots to estimate whether it would be more beneficial to purchase a long-term instance right away for future use.

In particular, looking back at the past interval $[t-\tau^l+2, t+1]$, there were VM demands served by short-term instances, according to decisions of **LTP** and **STP** (to be discussed in Alg. 3 shortly) made in the past interval. We run a subroutine **MinST** (with detailed steps given in [27] due to space limit) to retrospectively compute the optimal strategy for short-term VM purchase/renewal and deployment to cover these demands during this interval, which incurs the minimal cost (referred to as F in Alg. 2) based on full knowledge of the past interval. We compare this optimal cost F with the cost if we had deployed a long-term VM ($\beta_m + d_m$): if $F > \beta_m + d_m$, we know that we should have used a long-term VM to cover these demands that our online algorithm fulfilled using short-term instances, and then we will learn from the mistake and purchase a long-term instance right away.

In Alg. 2, we loop through all layers as shown in Fig. 2 (b) (demand not covered by long-term VMs yet) in a bottom-up

Algorithm 2: Online Algorithm for Purchasing Long-Term type-m VMs for $t+1$ – **LTP**

Input: t, $\mathbf{v}_m$, $\hat{\mathbf{y}}_m$, τ^l, τ^s, α_m, β_m, d_m
Output: $y_m(t+1)$, $\hat{\mathbf{y}}_m$
1 **while** $\sum_{k=t-\tau^l+2}^{t+1} v_m(k) - \hat{y}_m(k) > 0$ **do**
2 Compute $F = \mathbf{MinST}(\mathbf{v}_m, \hat{\mathbf{y}}_m, \tau^l, \tau^s, t, \alpha_m, d_m)$;
3 **if** $F \geq \beta_m + d_m$ **then**
4 Update $y_m(t+1) = y_m(t+1) + 1$;
5 $\hat{y}_m(k) = \hat{y}_m(k) + 1, \forall t - \tau^l + 2 \leq k \leq t + \tau^l$;
6 **else**
7 Break;/* $y_m(t+1)$ is ready*/
8 **end**
9 **end**
10 Purchase and launch $y_m(t+1)$ new long-term type-m instances;
11 Return $y_m(t+1)$, $\hat{\mathbf{y}}_m$;

Algorithm 3: Online Algorithm for Purchasing Short-Term Type-m VMs for $t+1$ – **STP**

Input: t, $v_m(t+1)$, $\hat{\mathbf{y}}_m$, $\mathbf{u}_m$, τ^l, τ^s, α_m, d_m
Output: $u_m(t+1)$
1 $\hat{u}_m(t+1) = \sum_{t-\tau^s+2 \leq k \leq t+1} u_m(k)$;
2 $R_m(t+1) = v_m(t+1) - \hat{y}_m(t+1) - \hat{u}_m(t+1)$;
3 **if** $R_m(t+1) \geq u_m(t+1-\tau^s)$ **then**
4 Renew all $u_m(t+1-\tau^s)$ instances that expire at t;
5 Purchase $R_m(t+1) - u_m(t+1-\tau^s)$ new short-term VMs;
6 $u_m(t+1) = R_m(t+1)$;
7 **end**
8 **if** $R_m(t+1) < u_m(t+1-\tau^s)$ **then**
9 Select $\max\{R_m(t+1), 0\}$ VMs, which have been idled for the least time, from set $\mathcal{U}_m(t+1-\tau^s)$ of short-term VMs expiring at t, and renew them;
10 $u_m(t+1) = \max\{R_m(t+1), 0\}$;
11 **foreach** *VM i in $\mathcal{U}_m(t+1-\tau^s)$ not selected above* **do**
12 **if** *i has been idled for less than $\lfloor \frac{d_m}{\alpha_m} \rfloor \tau^s$ time slots* **then**
13 Renew VM i;
14 $u_m(t+1) = u_m(t+1) + 1$;
15 **end**
16 **end**
17 **end**
18 Return $u_m(t+1)$;

fashion (the $while$ loop), and decide if a long-term instance should be purchased right away (increment $y_m(t+1)$ in line 4), based on retrospective examination of the strategies in the past τ^l time slots (lines 2-4). The algorithm stops purchasing long-term VMs when the optimal cost for using short-term instances (F) is lower than the cost for using a long-term instance, to cover the demand in a layer (line 7), *i.e.*, $F < \beta_m + d_m$ (which will be true for higher layers as well since higher layers have less demand), or all demands have been covered by long-term instances (line 1). $\hat{y}_m(k)$ is an auxiliary variable recording the number of layers in the demand figure, which have been examined for time slot k with a positive long-term VM purchase decision made. An algorithm illustration can be found in [27] due to space limit.

B. Online Algorithm for Purchasing Short-term VMs

If the available VMs (those purchased earlier plus new long-term VMs purchased in t) still cannot fulfil demand $v_m(t+1)$, short-term instances are purchased/renewed to cover the discrepancy. On the other hand, even if the available VMs can meet the demand, we may still wish to renew some expiring short-term VMs, preparing to serve demand that may arise in the future, to save deployment costs over the long run.

In **STP** in Alg. 3, we first compute the discrepancy $R_m(t+1)$ between demand ($v_m(t+1)$) and available long-term and short-term VMs ($\hat{y}_m(t+1) + \hat{u}_m(t+1)$) (lines 1-2). If the discrepancy is larger than the number of expiring short-term VMs in t (those purchased at $t+1-\tau^s$, line 3), we renew all these $u(t+1-\tau^s)$ VMs (line 4), and purchase additionally $R_m(t+1) - u_m(t+1-\tau^s)$ new short-term VMs from the cloud (line 5). If the discrepancy is less (line 8), we renew $R_m(t+1)$ (if > 0) out of the $u_m(t+1-\tau^s)$ expiring VMs (line 9) and examine all the other expiring VMs: if an expiring VM has been in use or has been idled for less than $\lfloor \frac{d_m}{\alpha_m} \rfloor \tau^s$ time slots, we renew this VM for future use (lines 11-15). The rationale is to use the break-even point $\lfloor \frac{d_m}{\alpha_m} \rfloor \tau^s$ to balance the tradeoff between retaining a short-term VM with per-timeslot cost α_m and deploying a new VM with deployment cost d_m.

C. Competitive Analysis of Online VM Purchase

We next temporarily ignore the online learning part in $POLAR$ and analyze the competitive ratio achieved by online decisions for VM purchase, $\mathbf{y}$ and $\mathbf{u}$, made in lines 16-17 of Alg. 1 by calling **LTP** and **STP**. Let $Cost(POLAR)$ denote the total cost for purchasing and deploying long-term and short-term VMs over T time slots in $POLAR$, based on VM demand $v_m(t), \forall t \in [T]$, predicted by online learning: $Cost(POLAR) = \sum_{t\in[T]} \sum_{m\in[M]} \alpha_m u_m(t) + \beta_m y_m(t) + d_m\big(y_m(t) + [u_m(t) - u_m(t-\tau^s)]^+\big)$ (objective value in (1) excluding the last term). Let $Cost(OPT)$ denote the minimum overall VM purchase and deployment cost computed by OPT, the offline optimal VM purchase solution derived with full knowledge of $v_m(t)$'s in the entire system lifetime – the same VM demands as predicted in $POLAR$. The competitive ratio is computed as $\frac{Cost(POLAR)}{Cost(OPT)}$.

Theorem 2. *POLAR in Alg. 1 is 3-competitive in overall VM purchase and deployment cost, as compared to the offline optimum derived under the same demand sequence $v_m(t), \forall m \in [M], t \in [T]$.*

D. Performance Analysis of the Complete $POLAR$ Algorithm

$POLAR$ combines an online learning module and an online VM purchase algorithm. Following the standard metrics, a regret bound is shown for the online learning approach (Sec. IV-D) and a competitive ratio is derived for the online algorithm (Sec. V-C). To demonstrate the performance of the complete algorithm, we construct a performance ratio which jointly measures the regret of our VNF demand prediction and the competitiveness of online VM purchase:

$\text{Ratio}^T(POLAR)$, the ratio of the expected overall cost in (1) incurred by $POLAR$ (over random realization of $x_n(t)$'s) to that incurred by an optimal strategy $SOPT$. For each n, $SOPT$ uses the best static predictor $\bar{\theta}'_n$ defined in (9) for all $t \in [T]$, draws $x_n(t)$ from the distribution in (7) using $\bar{\theta}'_n$, and computes optimal strategy for long-term and short-term VM purchase/renewal for any realization of $x_n(t)$'s, by solving (1) based on full knowledge of these $x_n(t)$'s in T. We have

$$\text{Ratio}^T(POLAR) = \frac{E_{x_n(t)}[G(POLAR)]}{E_{x_n(t)\sim\mathcal{D}_{x_n(t)}(\bar{\theta}'_n)}[G(SOPT)]}, \quad (11)$$

where $G(\cdot)$ denotes the value of the objective function in (1).

Theorem 3. *The performance ratio* (11) *of* $POLAR$ *in Alg. 1, is upper bounded by* $3\big(1 + \max_{m\in[M]} \frac{2s_m^{max}\chi\tau^l}{(\sqrt{2}-1)(\beta_m+d_m)\sqrt{T}}\big)$, *where* $s_m^{max} = \max_{t\in[T]} s_m(t)$ *and* $\chi = \max_{n\in[N]} \frac{\max_{t\in[T]} \sum_{j\in[J]:t_j^- \le t \le t_j^+, n\in\mathbb{L}^j} \hat{\lambda}_n^j(t)\pi_j(t)}{\min_{t\in[T]} \sum_{j\in[J]:t_j^- \le t \le t_j^+, n\in\mathbb{L}^j} \hat{\lambda}_n^j(t)\pi_j(t)}$.

A larger $\frac{s_m^{max}}{\beta_m+d_m}$ indicates higher VM cost in case of under-estimation of VM demands (such that backup VMs are used). A larger τ^l leads to a smaller lower-bound of the expected $G(SOPT)$. A larger χ comes with greater traffic rate fluctuation. All lead to a larger performance ratio. In addition, with the increase of T, $Ratio^T(POLAR)$ approaches 3, the competitive ratio of the online VM purchase algorithm given in Theorem 2. The implication is that influence of inaccurate demand prediction fades when T grows.

VI. Performance Evaluation

A. Simulation Setup

We simulate an NFV system lasting tens of thousands of time slots (T). Each time slot is 10 minutes long. We simulate service chains and their flow rates using Amazon Elastic MapReduce Trace [28]. Each record in the trace contains a HTTP request file, a source IP address, and the time stamp. For each time slot t, we group records containing the same IP address and a time stamp falling into the slot, by summing up their HTTP request files, and consider each group as one flow. Each flow goes through a service chain containing $2-5$ VNFs, randomly chosen among these given in the table below [29]. The traffic rate $\pi_j(t)$ injecting into a service chain is computed by dividing the total request file size by the duration of a time slot. Flow rate change ratios ($\lambda_n^j(t)$) along the service chain is randomly picked in $[0.5, 2]$. Over different time slots, the flows (obtained as above) with the same source IP address are considered to be the same flow with fluctuating flow rates over time. The arrival time of such a flow is the first time the source IP address appears in the trace, and the departure time is the last time that IP address appears. We simulate around 10^4 flows in total.

According to CPU requirement of VNFs in the table and simulating Amazon EC2 instances, we choose m4.xlarge, m4.xlarge, t2.medium and c4.2xlarge as the hosting VM types for the four VNFs, respectively. Each VM instance can be purchased as a 1-year-term reserved instance or an on-demand instance lasting one hour. Thus, $\tau^s = 6$ and $\tau^l = 52560$. α_m, β_m are set according to prices of these instances in Amazon EC2 [10]. d_m/α_m is set uniformly at random within $[0.1, 3]$. $s_m(t) = 2(\beta_m + d_m)$. All are default settings.

Network Function	CPU Required	Processing Capacity per Instance
Firewall	4	900Mbps
Proxy	4	900Mbps
NAT	2	900Mbps
IDS	8	600Mbps

B. Impact of Parameters

We first evaluate the impact of inaccurate estimation of x_n^{*max} on $Regret^T(POLAR)$ for online learning. We run Alg. 1 using the actual x_n^{*max} (derived under default setting of $\pi_j(t)$'s and $\lambda_n^j(t)$'s), $\frac{x_n^{*max}}{2}$ and $2x_n^{*max}$ in computing the prediction update factor η. Fig. 3 shows that both over-estimation and under-estimation only have minor influence on the regret. The regret under real-world input is much smaller than the theoretical upper-bound given in Theorem 1.

In Fig. 4, we multiply each $\pi_j(t)$ by a random number within $[3, 5]$ ($[0.1, 0.5]$) to obtain a larger (smaller) x_n^{*max}. A larger x_n^{*max} leads to higher regret, consistent with Theorem 1. We also evaluate the regret using $\eta = \frac{x_n^{*max}}{\sqrt{2T}}$ to update $\theta_n(t)$, and observe that regret by $POLAR$, using ϕ to compute η instead of T, is similar to the regret when using T.

We next evaluate $Ratio^T(POLAR)$ by dividing the expected cost incurred by $POLAR$ (repeating experiments for 50 times) by that by $SOPT$, obtained by computing the best static predictor $\bar{\theta}_n$ to realize $x_n(t)$ and then exactly solving (1) using a brute force approach. Fig. 5 shows that a larger τ^l leads to a larger ratio in general, but the impact is not as significant as what the theoretical ratio in Theorem 3 ($3(\frac{2\tau^l}{\sqrt{T}} + 1)$ in this experiment) implies. In Fig. 6, a larger $\frac{s_m^{max}}{\beta_m+d_m}$ brings a larger ratio, which is nevertheless much smaller than the theoretical ratio – at least $3(\frac{10^5}{\sqrt{T}} + 1)$ in our setting. The ratios decrease when T grows.

C. Comparison with Alternative Heuristics

We further compare $POLAR$ with an online learning algorithm, Follow-The-Leader (FTL) from [30], and a heuristic, $Adaptive$. FTL uses $\theta_n(t+1) = \text{argmin}_{\theta\ge 0} \sum_{k=1}^t f_{nk}(\theta)$, the best static predictor based on history, to update $\theta_n(t+1)$ each time. $Adaptive$ changes the step size η in line 12 of $POLAR$ to $\eta(t) = \frac{\max_{k\in[1,t]} x_n^*(k)}{\sqrt{2\phi}}$, where $\max_{k\in[1,t]} x_n^*(k)$ is an alternative to x_n^{*max}. Fig. 7 shows that $POLAR$ outperforms FTL and $Adaptive$.

We next compare $POLAR$ with two heuristics $SLTP$ and $NRSTP$, which differ in the online VM purchase algorithm. $SLTP$ simplifies LTP by setting F as the short-term VM purchase/renewal and deployment cost incurred by $POLAR$ during $[t-\tau^l+2, t+1]$. $NRSTP$ does not renew any expiring short-term instance if it is idle. These simpler heuristics sacrifice cost effectiveness, as shown in Fig. 8. We also observe that when the break-even point (decided by $\frac{d_m}{\alpha_m}$) is larger (more aggressively renewing idling short-term VMs), the ratio is larger, though it does not appear directly in theoretical ratio.

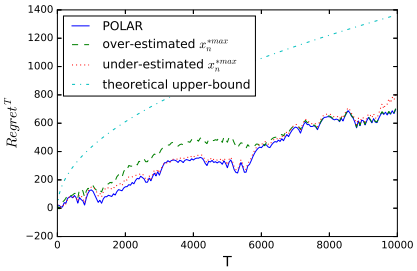

Fig. 3: $Regret^T$ (inaccurate x_n^{*max})

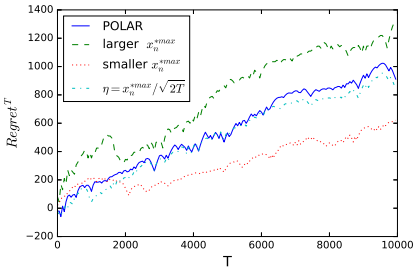

Fig. 4: $Regret^T$ (varying η)

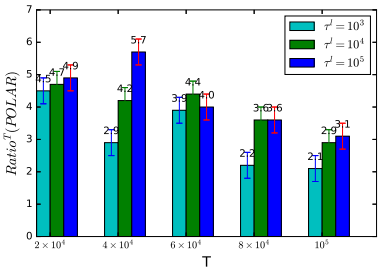

Fig. 5: $Ratio^T$ (varying τ^l)

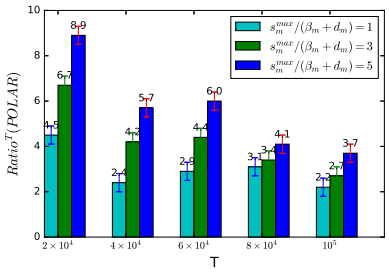

Fig. 6: $Ratio^T$ (varying $\frac{s_m^{max}}{\beta_m+d_m}$)

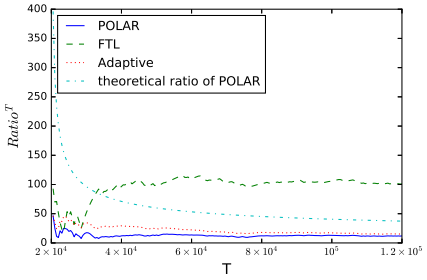

Fig. 7: $Ratio^T$ (different predictions)

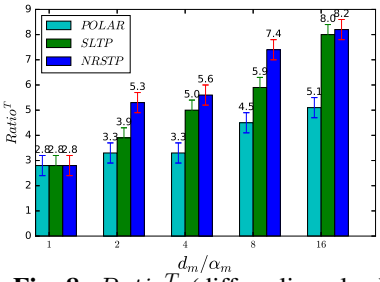

Fig. 8: $Ratio^T$ (diff. online alg.s)

VII. CONCLUSION

This paper studies a novel NFV brokerage service where an NFV provider acquires resources from the cloud to serve service chain demands of customers. For better flow service quality assurance and cost effectiveness, we adopt online learning to predict the upcoming traffic demand, and adjust VNF deployment in a proactive fashion. We design efficient online optimization techniques to purchase cloud VMs of different usage durations. The online VM purchase algorithm works in concert with online learning in a unified algorithm framework. We define a novel performance ratio that jointly measures sub-optimality of the prediction and the online algorithm, and demonstrate good performance of our algorithm in both theoretical analysis and trace-driven simulation.

VIII. ACKNOWLEDGEMENTS

This work was supported in part by grants from Hong Kong RGC under the contracts HKU 718513, 17204715, 17225516, 717812, C7036-15G (CRF), a grant from the Natural Sciences and Engineering Research Council of Canada (NSERC), a grant from Wedge Networks, and a grant NSFC 61628209.

REFERENCES

[1] "Network Functions Virtualization," https://goo.gl/q2uBgy.
[2] "Service Function Chaining Use Cases In Data Centers, IETF Draft," https://goo.gl/A1gpRm.
[3] "Network Functions Virtualisation Use Cases," http://goo.gl/1HrZlw.
[4] "TNOVA," http://www.t-nova.eu.
[5] "Network Function Virtualization: Rebuilding Network Functions and Open Architectures."
[6] "New NFV Vendor Ecosystem, Usual Suspects: Cisco, Juniper ALU, HP," http://goo.gl/XpCB9H.
[7] "NFV Market on Pace to Hit $11 Billion," http://goo.gl/5e8vDk.
[8] X. Wang, C. Wu, Z. Li, Z. L. Franck Le, Alex Liu, and F. Lau, "Online VNF Scaling," in *Proc. of IEEE Cloud*, 2016.
[9] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, "Elastic Virtual Network Function Placement," in *Proc. of IEEE CloudNet*, 2015.
[10] "EC2 - Amazon Web Services," https://aws.amazon.com/ec2/.
[11] "Microsoft Azure," https://azure.microsoft.com/.
[12] S. Shalev-Shwartz, "Online Learning and Online Convex Optimization," *Journal of Foundations and Trends in Machine Learning*, vol. 4, no. 2, pp. 107 – 194, 2012.
[13] M. Zinkevich, "Online Convex Programming and Generalized Infinitesimal Gradient Ascent," in *Proc. of International Conference on Machine Learning (ICML)*, 2003.
[14] N. Chen, A. Agarwal, A. Wierman, S. Barman, and L. L. H. Andrew, "Online Convex Optimization Using Predictions," in *Proc. of ACM SIGMETRICS*, 2015.
[15] N. Chen, J. Comden, Z. Liu, A. Gandhi, and A. Wierman, "Using Predictions in Online Optimization: Looking Forward with an Eye on the Past," in *Proc. of ACM SIGMETRICS*, 2016.
[16] H. Moens and F. De Turck, "VNF-P: A Model for Efficient Placement of Virtualized Network Functions," in *Proc. of International Conference on Network and Service Management (CNSM)*, 2014.
[17] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near Optimal Placement of Virtual Network Functions," in *Proc. of IEEE INFOCOM*, 2015.
[18] L. Zheng, C. Joe-Wong, C. G. Brinton, C. W. Tan, S. Ha, and M. Chiang, "On the Viability of a Cloud Virtual Service Provider," in *ACM SIGMETRICS*, 2016.
[19] W. Wang, B. Liang, and B. Li, "To Reserve or Not to Reserve: Optimal Online Multi-Instance Acquisition in IaaS Clouds," in *USENIX ICAC*, 2013.
[20] M. S. Manasse, "Ski Rental Problem," *Encyclopedia of Algorithms*, pp. 849–851, 2008.
[21] "Reserved Instances – AWS EC2," http://goo.gl/9s9i3g.
[22] "Provider Provisioned VPN Terminology," https://ietf.org/rfc/rfc4026.
[23] "Benchmarking Terminology for Firewall Performance," https://ietf.org/rfc/rfc2647.
[24] "Bro Cluster," https://www.bro.org/sphinx-git/cluster/index.html.
[25] "AWS Developer Forums: Auto-renewal of reserved instances?" https://forums.aws.amazon.com/message.jspa?messageID=209350.
[26] W. Stallings, *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud*. Addison-Wesley Professional, 2015.
[27] "Proactive VNF Provisioning with Multi-timescale Cloud Resources," Tech. Rep., `https://goo.gl/6w1oiq` 2016.
[28] "Amazon EMR Trace," s3://us-east-1.elasticmapreduce.samples.
[29] M. Faizul Bari, S. Rahman Chowdhury, R. Ahmed, and R. Boutaba, "On Orchestrating Virtual Network Functions in NFV," *ArXiv e-prints*, Mar. 2015.
[30] S. D. Rooij, T. V. Erven, P. D. Grnwald, and W. M. Koolen, "Follow the Leader If You Can, Hedge If You Must," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1281 – 1316, 2014.